Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Peter Dolenec

Identifikacija in overjanje

magistrsko delo

študijski program druge stopnje
računalništvo in informatika

Mentor: prof. dr. Aleksandar Jurišić

Ljubljana, 2017

Zahvaljujem se svojemu mentorju prof. dr. Aleksandru Jurišiću, ki me je med pisanjem magistrskega dela spodbujal ter za vso njegovo pomoč in nasvete. Prav tako se zahvaljujem vsem, ki so mi stali ob strani in me podpirali v času študija.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
AES	Advanced Encryption Standard	Simetrična šifra, izbrana na NIST izboru leta 2000 (nadomesti DES)
API	Application Programming Interface	Programski vmesnik
CBC	Cipher Block Chaining	Verižni način delovanja bločne šifre
DES	Data Encryption Standard	Najbolj znana bločna simetrična šifra
ECC	Elliptic Curve Cryptosystem	Kriptosistem z eliptičnimi krivuljami
ECDSA	Elliptic Curve Digital Signature Algorithm	Algoritem za digitalni podpis z eliptičnimi krivuljami
FIPS	Federal Information Processing Standard	Ameriški zvezni urad za standarde
HSM	Hardware Security Module	Strojni varnostni modul
KDF	Key Derivation Functions	Funkcije za izpeljavo ključa
MD	Message Digest	Zgostitev
NIST	National Institute of Standards and Technology	Ameriški zvezni urad za standarde in tehnologijo
NSA	National Security Agency	Agencija za nacionalno varnost
PBKDF	Password Based Key Derivation Function	Funkcija za izpeljavo gesla iz ključa

PIN	Personal Identification Number	Osebna identifikacijska številka
SHA	Secure Hash Algorithm	Varni zgoščevalni algoritem
SIG	Special Interest Group	Skupina, ki skrbi za razvoj bluetooth standardov in protokolov
SSL	Secure Socket Layer	Varnost prenosnega nivoja

Povzetek

Dan danes na vsakem koraku uporabljamo gesla za prijavo v različne sisteme. Če želimo zadostiti nekim minimalnim varnostnim zahtevam, moramo za vsak sistem (spletno stran) uporabljati drugo geslo, ki pa mora biti tudi dovolj kompleksno. Vsa ta gesla pa si je praktično nemogoče zapomniti, zato rabimo posebne aplikacije, ki skrbijo za njihovo varno hranjenje.

Najprej si bomo pogledali osnove overjanja. Pri tem bomo predstavili različne načine overjanja (z gesli, biometričnimi podatki ter varnostnimi žetoni) in njihove prednosti ter slabosti. Opisali bomo nekaj aplikacij, ki rešujejo problem upravljanja z gesli ter njihove pristope.

Glavni del magistrske naloge opisuje izdelavo aplikacije za hranjenje gesel. Aplikacija je sestavljena iz dveh delov. Prvi del predstavlja mobilna aplikacija in je namenjena hranjenju gesel, drugi del pa je aplikacija namenjena uporabi na računalniku in se uporablja za dostop do shranjenih gesel.

Ključne besede: gesla, upravljanje z gesli, varnost gesel, simetrična kriptografija, IJK.

Abstract

We use passwords for logging in different systems on a daily basis. If we want to satisfy some minimal security requirements, we need to use a different password for every system (web page) and this password must also be sufficiently complex. It is hard to remember so many passwords. That's why we need special applications for safely storing them.

First we will present the basic of authentication. We will start by going through different kinds of authentications (with passwords, biometric data and security tokens) and present their advantages and weaknesses. We will also present a few applications that deal with this problem.

The main part describes the development of our application for storing passwords. The application consists of two parts. The first part is mobile application and is intended for storing passwords. The second part is an application designed for a computer and is used for accessing stored passwords.

Keywords: passwords, passwords management, passwords safety, symmetric cryptography, PKI.

Poglavje 1

Uvod

V današnjem času skoraj na vsakem koraku uporabljamo gesla za prijavo v različne sisteme (npr. pametne telefone, računalnike, razne aplikacije, kot je elektronska pošta, itd.). Če želimo zadostiti vsaj minimalnim varnostnim zahtevam, potrebujemo drugačno geslo za vsak sistem. Le-ta morajo biti dovolj kompleksna. Ker si je težko zapomniti vsa ta gesla, so si jih uporabniki najprej pričeli zapisovati na kakšen list papirja ali pa v tekstovno datoteko. Ker lahko tak način hranjenja gesel občutno zmanjša njihovo varnost, so se pričeli uporabljati posebni programi za njihovo hranjene (t.i. aplikacija za hranjenje ključev, ang. key manager). Glede na to, da ima skoraj vsak uporabnik več kot eno napravo na kateri uporablja gesla, se morajo ti podatki nekako med seboj sinhronizirati. Najbolj pogost način sinhronizacije podatkov je z uporabo storitev v oblaku. To pomeni, da so vsa gesla shranjena v oblaku in potem lahko iz vseh naprav dostopamo do njih. Na prvi pogled je to idealna rešitev, toda s selitvijo podatkov v oblak izgubljamo nadzor na njimi in tako je potrebno 100% zaupati ponudniku, da ne bo prišlo do kakšnih zlorab. Hkrati je taka zbirka idealna tarča za morebitnega napadalca, ki bi z vdorom do teh podatkov imel dostop do vseh gesel uporabnikov.

Kot alternativa geslom se v zadnjem času za potrebe overjanja vedno bolj uveljavlja uporaba biometričnih podatkov. Zasluga gre predvsem izboljšavi in pocenitvi ustrezne strojne opreme ter njeno vgradnjo v mobilne naprave.

Tako ima skoraj vsak nov pametni mobilni telefon vgrajen vsaj en tak senzor. Najbolj pogosto se uporablja čitalec prstnih odtisov. Uporaba biometričnih podatkov je za končne uporabnike precej bolj preprosta, kot uporaba gesel, saj imajo npr. prstni odtis vedno pri sebi in ga ne morejo pozabiti. Poleg tega je celoten postopek overjanja tako hiter, da ga običajno uporabnik niti ne opazi.

V tej nalogi predlagamo rešitev, ki bi omogočala, da uporabnik do gesel dostopa iz vseh svojih naprav, hkrati pa so gesla varno shranjena pri uporabniku in so vedno pod njegovim nadzorom. Razvili smo konceptno aplikacijo, ki je sestavljena iz dveh delov. Prvi (strežniški) del hrani gesla na mobilnem telefonu in se preko brezžične povezave bluetooth poveže z računalnikom, na katerem je nameščen drugi del aplikacije (odjemalec). Ker brezžična povezava, sama po sebi, ne zagotavlja dovolj visoke stopnje varnosti, smo zanjo poskrbeli na podatkovnem nivoju.

V naslednjem poglavju bomo najprej predstavili osnove overjanja. V tretjem poglavju si bomo nekoliko podrobneje ogledali nekaj rešitev (aplikacij) za hranjenje gesel ter alternativne pristope k overjanju. Sledi opis uporabljenih (predvsem kriptografskih) tehnologij. V petem poglavju bomo podrobneje predstavili našo aplikacijo. Sklepne misli so podane v šestem poglavju.

Poglavje 2

Osnove overjanja

Za opis postopka preverjanja, kdo je uporabnik, sledimo novejši učbenik *Security in Computing* avtorjev Charles in Shari Lawrence Pfleeger [9, Sec. 2.1]. Sestavljen je iz dveh delov:

(a) **identifikacije** – tj. postopek v katerem se uporabnik predstavi (npr. z imenom in priimkom ali vizitko, v primeru uradnega postopka pa z osebnim dokumentom) in

(b) **overjanja** – tj. postopek, v katerem se preveri, ali je identiteta iz prejšnje točke pravilna, z drugimi besedami, preveri, da je oseba res tista, za katero se je predstavila (uradna oseba običajno primerja sliko na osebnem dokumentu z osebo, prodajalec prosi za podpis, ki ga nato primerja s tistim na dokumentu,...).

Koncepta identifikacije in overjanja ljudje pogosto zamenjujemo. Podatki o identiteti, npr. ime, so običajno javni in nezaščiteni. Po drugi strani overjanje potrebuje določeno stopnjo zaščite. Če je neka identiteta javna, jo lahko vsak uporabi za lažno predstavitev. Dokazovanje z overjanjem nam omogoča, da z večjo verjetnostjo ločimo med tistimi, ki se pretvarjajo in pravo osebo.

Na področju računalništva je overjanje najbolj pogosto proces, pri katerem se mora strežnik prepričati, da je uporabnik zares tisti za kogar se

predstavlja. Zgled overjanja je vnos uporabniškega imena in gesla npr. pri prijavi v sistem.

Identiteta je pogosto dobro poznana, predvidljiva ali pa jo je možno uganiti. Ko uporabnik pošlje elektronsko sporočilo, je poleg pripet tudi njegov elektronski naslov. Številka bančnega računa je natisnjena na uporabnikovih čekih. Številka računa kreditne kartice je napisana na sami kreditni kartici. V vsakem od teh primerov uporabnik razkrije del svoje identitete. Pogosto se za uporabniško ime uporablja elektronski naslov uporabnika. Ker so ti podatki bolj ali manj javno znani, jih lahko vsak, ki jih pozna, uporabi in se z njimi predstavlja. Zato se smatra, da je overjanje bolj zanesljivo. Če poenostavimo, je identifikacija razkritje uporabnikove identitete, overjanje pa je postopek (za nas bo pogosto kriptografski protokol), pri katerem uporabnik želi potrditi, da je res tisti, za kogar se predstavlja. Če postopek overjanja ni dovolj učinkovit, bomo smatrali, da ni dovolj varen. Le-ta uporablja podmnožico naslednjih treh lastnosti za potrjevanje uporabnikove identitete:

(a) **nekaj kar uporabnik ve** – npr. gesla, številke (PIN) ali skrivno rokovanje,

(b) **nekaj kar uporabnik je** – to so biometrični podatki, ki temeljijo na fizičnih lastnostih uporabnika, kot so prstni odtis, šarenica, obraz, ali način hoje,

(c) **nekaj kar uporabnik ima** – t.i. varnostni žetoni, kot so fizični ključi, osebna izkaznica, vozniško dovoljenje, banča kartica itd.

V primeru plačila z bančno kartico se npr. običajno uporabljata dve lastnosti: *bančna kartica* (nekaj kar uporabnik ima) ter *PIN številka* (nekaj kar uporabnik ve).

2.1 Gesla

Uporaba gesel je v osnovi dokaj enostavna. Uporabnik mora najprej vnesti podatek o svoji identiteti (npr. uporabniško ime ali elektronski naslov). Ta

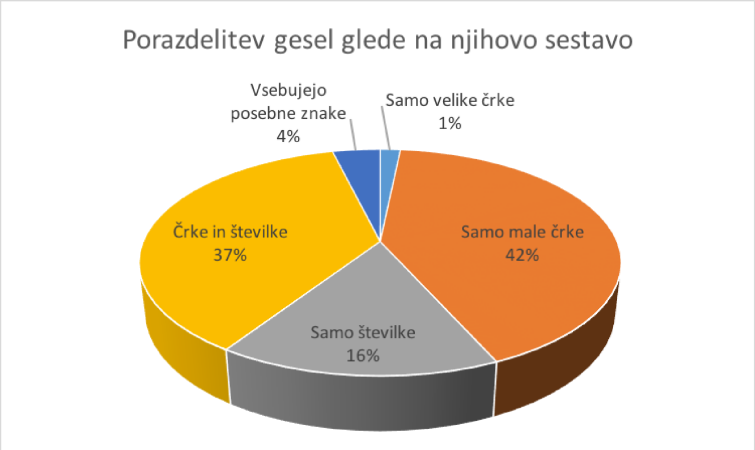
podatek je javno znan ali ga je možno dokaj enostavno uganiti tako, da ne zagotavlja neke zaščite. Nato mora vnesti svoje geslo. Če se geslo ujema s shranjenim geslom oz. njegovo zgostitvijo, se uporabniku dovoli dostop. Sicer sistem zahteva ponoven vpis gesla, ker je možno, da je uporabnik narobe vpisal svoje podatke ali naredil napako pri vnosu gesla. Očutljivejši sistemi lahko v primeru napačnega gesla časovno omejijo dostop (npr. če uporabnik 5-krat narobe vnese geslo, lahko ponovno poskusi šele čez 30 minut) oz. se lahko blokira uporabnikov račun in za deblokado mora iti čez drug postopek overjanja, ki je bolj temeljit in zahteva več osebnih podatkov (npr. varnostna vprašanja).

2.1.1 Statistika in varnost gesel

Uporaba gesel se smatra za relativno varen način overjanja, v primeru, če so le-ta dovolj dobra. Vendar so raziskave navad uporabnikov pokazale, da pogosto uporabljajo kratka gesla (30% gesel je krajših od sedem znakov, natančnejša razdelitev je vidna na Sliki 2.1). Nadalje 40% uporabnikov uporablja gesla, ki vsebujejo samo male črke (s tem v osnovi ni nič narobe, če le je geslo dovolj dolgo), kar 15% pa gesla, ki so sestavljena samo iz številk (podrobnejša porazdelitev je vidna na Sliki 2.2) [34, 65]. Glavni razlog za to je predvsem dejstvo, da si je težko zapomniti večje število različnih gesel. Da je stanje za uporabnika še slabše, določeni sistemi zahtevajo redno menjavo gesel (oz. staranje gesel). Tako je poznan primer, ko je nek sistem zahteval menjavo gesla vsake 3 mesece in ni dovoljeval ponovne uporabe istega gesla (po štiriindvajsetih spremembah se je prvotno geslo lahko ponovilo). Zato je nek uporabnik vsake tri mesece 24-krat zamenjal geslo, tako da je prišel nazaj na geslo, ki ga je predhodno uporabljal [9, Sec. 2.1]. Pogosto uporabniki v takem primeru, obstoječemu geslu samo dodajo neko številko (ali pa obstoječo št. povečajo za 1).

Zadnjih nekaj let podjetje SplashData objavlja seznam najbolj pogosto uporabljenih gesel. Tako čez celo leto zbirajo javno objavljene sezname gesel. Take sezname običajno najprej objavljajo hekerji ob uspešno izvedenih

Slika 2.1: Porazdelitev dolžine gesel.



Slika 2.2: Porazdelitev gesel glede na nabor uporabljenih znakov.

napadih na razne spletne strani. Konec leta objavijo seznam 25 najbolj pogosto uporabljenih gesel. Iz Tabele 2.1 je razvidno, da sta zadnja leta najbolj pogosti gesli "123456" ter "password".

Poleg tega, da uporabniki izbirajo slaba gesla, ne vedo, kako je za njihovo varnost poskrbljeno na posamezni spletni strani. Tako obstajajo spletne strani, ki gesla hranijo brez kakršnekoli zaščite (kot navadno besedilo), kar pomeni, da se lahko potencialni napadalec veliko lažje dokoplje do gesel, kot v primeru, ko se za hranjenje gesel uporabljajo zgostitve gesel (še bolje je, če se pri zgostitvah uporablja tudi sol) [32]. Zaradi tega moramo, če želimo zagotoviti vsaj minimalnim varnostnim zahtevam, za vsako spletno stran uporabiti drugo geslo. To pomeni, da bi moral imeti vsak uporabnik nekaj deset različnih gesel, kar je v praksi skoraj nemogoče. Po nekaterih raziskavah, okrog 60% uporabnikov uporablja ista gesla na več različnih straneh [22]. Posledično lahko napadalec ukrade geslo uporabnika na spletni strani s slabo zaščito in s tem pridobi dostop tudi do njegovih drugih računov, kjer uporablja isto oz. zelo podobno geslo.

2.1.2 Napadi na gesla

Obstaja več različnih napadov na gesla. V nadaljevanju bomo predstavili nekaj najbolj pogostih.

Napad s slovarjem

Napad s slovarjem je dokaj enostaven in hiter. Na spletu obstaja veliko aplikacij, ki omogočajo tovrstne napade. Ena takšnih (brezplačnih) aplikacij je John the Ripper [51]. Na voljo so tudi razna forenzična orodja, ki prav tako omogočajo različne napade na gesla. Primer take aplikacije je Passware Kit Forensic [50]. Poleg tega je na spletu možno najti tudi različne, že pripravljene slovarje. Tako lahko napadalec uporabi npr. slovar sestavljen iz najbolj pogostih gesel. Obstajajo tudi aplikacije, ki lahko iz neke spletne strani avtomatsko izdelajo slovar. To je lahko uporabno predvsem, ko napadalec želi izdelati slovar za specifično osebo. Recimo, da napadalec ve

#	2015	2014	2013	2012	2011
1	123456	123456	123456	password	password
2	password	password	password	123456	123456
3	12345678	12345	12345678	12345678	12345678
4	qwerty	12345678	qwerty	abc123	qwerty
5	12345	qwerty	abc123	qwerty	abc123
6	123456789	1234567890	123456789	monkey	monkey
7	football	1234	111111	letmein	1234567
8	1234	baseball	1234567	dragon	letmein
9	1234567	dragon	iloveyou	111111	trustno1
10	baseball	football	adobe123	baseball	dragon
11	welcome	1234567	123123	iloveyou	baseball
12	1234567890	monkey	admin	trustno1	111111
13	abc123	letmein	1234567890	1234567	iloveyou
14	111111	abc123	letmein	sunshine	master
15	1qaz2wsx	111111	photoshop	master	sunshine
16	dragon	mustang	1234	123123	ashley
17	master	access	monkey	welcome	bailey
18	monkey	shadow	shadow	shadow	passw0rd
19	letmein	master	sunshine	ashley	shadow
20	login	michael	12345	football	123123
21	princess	superman	password1	jesus	654321
22	qwertyuiop	696969	princess	michael	superman
23	solo	123123	azerty	ninja	qazwsx
24	passw0rd	batman	trustno1	mustang	michael
25	starwars	trustno1	000000	password1	football

Tabela 2.1: Seznam 25 najbolj pogosto uporabljenih gesel v zadnjih petih letih [59].

(npr. z uporabo socialnih omrežij), da je oseba navijač nekega nogometnega kluba, tako lahko uporabi tako aplikacijo za izdelavo manjšega slovarja iz besed uporabljenih na spletni strani tega kluba.

Za slovar se lahko uporabi tudi kakšen od slovarjev, ki se uporabljajo za preverjanje pravilnosti napisanih besed v določenem urejevalniku besedila. Takšni slovarji so prav tako dostopni na spletu. Naprednejše aplikacije za napad s slovarjem omogočajo tudi spremembe na besedah, ki so v slovarju. Tako lahko pred/za besedo dodajo poseben znak ali kakšno številko, poljubno spreminjajo male in velike črke, omogočajo zamenjavo črk s številkami, združevanje več besed iz slovarja ter na tak način izboljšajo možnosti za uspešen napad.

Napad z mavričnimi tabelami

Kadar ima napadalec dostop do zgostitev gesel, lahko uporabimo t.i. napad z mavričnimi tabelami. V njih so zapisana gesla ter njim pripadajoče zgostitve. Tako mora napadalec samo primerjati zgostitev gesla z zgostitvijo v tabeli in ko najde ujemanje, s tem dobi, tudi besedo, ki pripada tej zgostitvi. Mavrične tabele lahko napadalec prenese iz spleta ali jih sam izračuna.

Dober način za zaščito pred takim napadom je uporaba t.i. soli. To pomeni, da se pri izračunu zgostitve geslu doda še nek niz, ki mu pravimo sol. Le-ta običajno ni javno znana, vendar ne predstavlja nekega večjega varnostnega tveganja, če bi ta podatek prišel v javnost. Poleg tega obstajajo tudi različni načini, kako se sol doda geslu, npr. lahko se doda pred geslo (*sol + geslo*), lahko se doda za geslo (*geslo + sol*), lahko se naredi tudi xor operacija (*geslo xor sol*), ...

Če se uporablja sol, mora napadalec sam izračunati svojo mavrično tabelo. Poleg tega mora najprej ugotoviti, kaj se uporablja za sol ter kako se ta sol doda geslu. Pri tem je pomembno tudi ali vsa gesla v bazi uporabljajo isto sol ali se ta razlikuje od gesla do gesla. Če se za vsa gesla uporablja ista sol, lahko napadalec z enkratnim izračunom mavrične tabele uspešno pridobi vsa gesla. Drugače mora za vsako geslo izdelati svojo mavrično tabelo, kar

je pogosto časovno preveč potratno, da bi se napadalcu izplačalo.

Napad z grobo silo

Pri tem napadu napadalec poskuša vse možne permutacije črk, številk ter posebnih znakov. Ta metoda je (vsaj v teoriji) 100% uspešna. Težava je v časovni zahtevnosti, saj se le ta povečuje eksponentno glede na dolžino gesla. Če za primer vzamemo napad na geslo dolžine 1 do 8 znakov angleške abecede (vzamemo samo male črke) imamo

$$26^1 + 26^2 + \cdots + 26^8 = (26^8 - 1)/25$$

možnih gesel, ter predpostavimo, da za preverjanje uporabimo računalnik in za vsako geslo porabimo eno milisekundo, bi nam to vzelo približno 150 let. Če bi pospešili postopek preverjanja na eno mikrosekundo za vsako geslo, bi napad trajal približno le dva meseca. Slednje je za napadalca, lahko že sprejemljivo. Pri tem je potrebno poudariti, da je ta napad možno dokaj enostavno paralerizirati in uporabiti računsko moč grafičnih kartic ter tako pohitriti celoten proces. Z uporabo računalništva v oblaku je praktično vsakomur omogočen dostop do precejšnje računske moči za relativno nizko ceno. Kljub temu smo v primeru uporabe nekoliko daljšega (npr. 12 znakov) ter dovolj kompleksnega gesla (male in velike črke, številke, posebni znaki) varni pred tem napadom, ker bi zanj potrebovali preveč časa oz. računske moči.

Socialno inženirstvo

Namesto, da bi napadalec iskal napake v sistemu oz. poskušal ugotoviti uporabnikovo geslo, se le ta osredotoči na uporabnika. Pri tem napadu poskušamo z manipulacijo oz. zlorabo zaupanja pridobiti željene podatke. Obstaja več tehnik socialnega inženiringa. Nekatere so bolj tehnične narave npr. spletno ribarjenje, kjer napadalec postavi lažno spletno stran v katero uporabnik vnese svoje podatke. Druge so bolj sociološke oz. psihološke npr. v preteklosti so bili primeri, ko so napadalci klicali uporabnike in jim rekli,

da kličejo iz banke in da rabijo njihovo številko PIN. Veliko uporabnikov jim je povedalo svojo PIN številko, saj so verjeli, da jih resnično kličejo iz banke.

2.1.3 Močna gesla

Pri večini napadov na gesla smo odvisni predvsem od tega, koliko časa lahko napadalec nameni za uspešen napad. Zato je zelo pomembna dolžina gesla ter njegov nabor znakov. Uporaba velikih in malih črk, številk ter posebnih znakov precej izboljša kvaliteto gesel. Za napad z grobo silo na geslo dolžine 6 znakov, ki je sestavljeno samo iz npr. malih črk bi potrebovali približno 100 ur, če pa bi uporabili tudi velike črke in številke, bi potrebovali približno dve leti. Tako je priporočljivo, da se uporablja daljša gesla, ki vsebujejo male in velike črke, številke ter posebne znake (v Tabeli 2.2 se dobro vidi eksponentna rast vseh možnosti glede na dolžino gesla). Vendar si je taka gesla težko zapomniti, zato je priporočljivo uporabljati posebne aplikacije, ki so namenjene hranjenju gesel, hkrati pa omogočajo tudi generiranje varnih gesel. Če si želimo geslo lažje zapomniti, lahko uporabljamo tudi kakšne besedne zveze. Pri tem smo res nekoliko bolj izpostavljeni napadom s slovarjem, vendar v primeru dovolj dolgega gesla tudi tak napad ni učinkovit. Če za predstavitev zahtevnosti vzamemo angleški slovar, ki se uporablja za preverjanje pravopisa v urejevalnikih besedila, ta vsebuje približno 80.000 besed in si izberemo geslo sestavljeno iz štirih besed dobimo $80.000^4 \approx 4 * 10^{19}$ različnih gesel.

V preteklih mesecih je prišla v javnost novica, da so hekerji dvakrat vdrli v sistem podjetja Yahoo ter jim ukradli podatke. Prvi napad se je zgodil avgusta 2013. Takrat so ukradli podatke več kot milijardi uporabnikov [62]. Leta 2014 pa so ukradli podatke okoli 500 milijonov uporabnikov [63]. Med ukradenimi podatki so imena, naslovi elektronske pošte, telefonske številke, datumi rojstva, zgostitve gesel in v nekaterih primerih tudi šifrirana ali nešifrirana varnostna vprašanja in odgovori [61]. Pri tem je potrebno omeniti, da so za izračun zgostitev gesel uporabljali zgoščevalno funkcijo MD5, ki je že dalj časa neprimerna za uporabo[1]. Poleg tega pa je tudi zaskrbljujoče,

[1]Leta 2004 so raziskovalci dokazali, da MD5 ni odporen na trke. Do leta 2007 so odkrili

dolžina	samo male črke	vsi znaki na tipkovnici
3	17.576	857.375
4	456.976	81.450.625
5	11.881.376	7.7337.809.375
6	308.915.776	735.091.890.625
7	8.031.810.176	69.833.729.609.375
8	208.827.064.576	6.634.204.312.890.620
9	5.429.503.678.976	630.249.409.724.609.000
10	141.1677.095.653.376	59.873.693.923.837.900.000
11	3.670.344.486.987.780	5.688.000.922.764.600.000.000
12	95.428.956.661.682.200	540.360.087.662.637.000.000.000
13	2.481.152.173.203.740.000	51.334.208.327.950.500.000.000.000
14	64.509.974.703.297.200.000	4.876.749.791.155.300.000.000.000.000
15	1.677.259.342.285.730.000.000	463.291.230.159.753.000.000.000.000.000

Tabela 2.2: Število možnih permutacij glede na dolžino gesla.

da so podatki o napadih prišli v javnost precej pozno (septembra in decembra 2016), tako uporabniki niso vedeli, da bi morali zamenjati svoja gesla.

2.2 Biometrični podatki

V zadnjem času se je močno povečala uporaba biometričnih podatkov za potrebe overjanja uporabnikov. To gre pripisati predvsem pocenitvi ustrezne strojne opreme ter njihov vgradnji v mobilne naprave. Tako ima velika večina novih pametnih telefonov vgrajen vsaj en čitalec namenjen posebej za preverjanje uporabnikovih biometričnih podatkov. Najbolj pogosto je to čitalec prstnih odtisov.

Uporaba biometričnih podatkov je za končnega uporabnika verjetno res

še več pomanjkljivosti. Leta 2010 je Software Engineering Institute izjavil, da je "MD5 algoritem zlomljen in neprimeren za nadaljnjo uporabo"[46].

bolj preprosta, saj jih imajo vedno pri sebi in jih ni mogoče pozabiti. V večini primerov je uporaba precej hitrejša, kot vnašanje gesel.

Biometrični podatki so biološke lastnosti, ki temeljijo na fizičnih lastnostih človeškega telesa. Če naštejemo samo nekaj lastnosti:

- prstni odtis,
- deli očesa (mrežnica, šarenica),
- glas,
- rokopis,
- obraz,
- geometrija roke,
- vzorec ven,
- zapis zob,
- RNK (DNA),
- hoja
- itd.

Overjanje z biometričnimi podatki ima svoje prednosti (v primerjavi z gesli), saj jih ni mogoče izgubiti, ukrasti, pozabiti ali jih deliti z drugimi, so pa vedno na voljo. Te karakteristike je težko, če ne nemogoče, ponarediti. Vendar ima uporaba biometričnih podatkov tudi nekaj težav:

(a) Uporaba biometričnih podatkov je relativno nova in nekaterim ljudem se zdi njihova uporaba precej vsiljiva. Tako so lahko ljudje v nekaterih državah užaljeni, če morajo dati svoje prstne odtise, saj imajo občutek, da jih obravnavajo kot kriminalce.

(b) Uporaba biometričnih čitalcev in njihova primerjava lahko postanejo ena točka odpovedi (ang. single point of failure). Za primer vzamemo sistem, ki za overjanje pri plačilu uporablja biometrične podatke. Če se pojavi težava pri plačilu s kreditno kartico lahko uporabnik uporabi

drugo kreditno kartico. Če pa sistem ne prepozna njegovega prstnega odtisa, ne more uporabiti drugega. Prepoznava prstnega odtisa je vezana na en prst, saj vzorec enega ni enak vzorcu drugega prsta.

(c) Biometričnih podatkov ne moremo spreminjati, saj so odvisni od naših fizičnih lastnosti. Če si za primer vzamemo prstni odtis, je jasno, da ga puščamo za seboj na mnogih površinah, ki se jih dotaknemo. Tako ga napadalec relativno enostavno pridobi in nato uporabi za lažno identifikacijo (ang. impersonation). Vendar je pomembno poudariti, da mora imeti v tem primeru napadalec dostop do potencialne "tarče" (kar pa pogosto ni možno). Seveda je dopustna tudi uporaba več različnih biometričnih testov (npr. prstni odtis, prepoznava glasu in prepoznava obraza), da na ta način napadalcu dodatno otežimo delo.

(d) Resen problem je tudi, da se pojavi vprašanje kaj, če uporabnik doživi neko nesrečo in se zaradi tega biometrični podatki spremenijo. Kako ukrepati v takih primerih. Recimo, da neka oseba za odklepanje vhodnih vrat svojega stanovanja uporablja prepoznavo glasu, potem pa se nekega dne zgodi, da ga zaradi prehlada, sistem ne prepozna, zna biti to precej nerodno. V takih primerih se običajno uporabi rezerven način overjanja (v danem primeru fizičen ključ za vrata).

Tukaj naletimo na novo težavo. V svetu računalništva je običajno rezerven način overjanja neko daljše geslo (npr. PUK, potem ko smo trikrat vnesli napačen PIN). Tu pa smo zopet na začetku zgodbe, saj moramo sedaj hraniti še dodatno/daljše geslo in je težava s hranjenjem gesel še večja. V primeru, če geslo ni dovolj varno, predstavlja še dodatno varnostno tveganje, saj je za napadalca dovolj, da uspešno napade najšibkejši člen. Opozorimo tudi na problem, da si gesel, ki jih redko uporabljamo (npr. zaradi pogoste uporabe biometrike) težje zapomnimo.

(e) Pri uporabi biometričnih podatkov prihaja tudi do napačnih negativnih ujemanj in napačnih pozitivnih ujemanj. Za primer vzemimo upo-

rabo prstnega odtisa za dostop do šifriranih podatkov na trdem disku. Vsakič, ko poteka branje prstnega odtisa, se primerjajo podatki s shranjenimi podatki in primerjava vrne nek odstotek ujemanja. Tako sistem odobri dostop uporabniku, če je ujemanje dovolj visoko (npr. 95%). Pri tem postopku pa včasih pride tudi do napačnih negativnih ujemanj. V tem primeru mora uporabnik ponoviti postopek. Možno pa je to izkoristiti tudi sebi v prid. Če vzamemo za primer identifikacijo oseb s prepoznavo roženice. Z uporabo barvnih leč je možno sistem pretentati, da pride do napačnega negativnega ujemanja. Tako bi lahko nek iskani kriminalec uporabil ta način, da ga tak sistem ne bi prepoznal. Običajno je bolj nevarno, kadar pride do napačnih pozitivnih ujemanj, kar pomeni, da je sistem odobril dostop uporabniku, ki za to nima dovoljenja.

(f) Dodatna težava biometričnih podatkov je tudi način njihove uporabe. Ko se potrdi ujemanje, dobimo dostop do šifrirnega ključa, s katerim nato dešifriramo podatke na disku. Tukaj se pojavi nova težava, saj dostop do šifrirnega ključa ni pogojen z nekimi kriptografskimi omejitvami, ampak samo z neko vnaprej določeno verjetnostjo. Tukaj je precej odvisno od same implementacije sistema in do česa ima napadalec dostop. Tako lahko poskuša vplivati na sistem, da zahteva manjši odstotek ujemanja, kot je bil v osnovi predviden ali pa poskuša direktno dostopati do šifrirnega ključa. Če na drugi strani to primerjamo z uporabo gesel, je zgodba precej drugačna. Z uporabo ustrezne zgoščevalne funkcije lahko zagotovimo relativno varno in sto odstotno ujemanje vpisanega gesla s shranjenim geslom (shranjena je zgostitev gesla, pri tem je potrebno poudariti, da se lahko dve različni gesli preslikata v isto zgostitev, zato je zelo pomembno, da se uporablja čim boljša zgoščevalna funkcija). Po potrditvi ujemanja pa lahko z uporabo funkcije, ki iz danega gesla naredi novo geslo (lahko bi uporabili tudi zgoščevalno funkcijo), generiramo šifrirni ključ, s katerim dešifriramo podatke. Ker šifrirni ključ ni nikjer shranjen, je tak način precej bolj varen, ven-

dar samo, če uporabljamo dovolj dobro geslo in ustrezne kriptografske funkcije.

Uporabo biometričnih podatkov in gesel si zelo različno razlagajo tudi posamezne zakonodaje (predvsem v primeru kazenskega pregona). Tako je leta 2014 okrožno sodišče v Virginiji odločilo, da posameznik v kazenskem postopku ni dolžan povedati gesla, s katerim je zaklenjen njegov pametni telefon, ker bi s tem kršili klavzulo samoobtožbe [17]. Hkrati pa so odločili, da je posameznik lahko prisiljen, dati svoj prstni odtis za potrebe odklepanja pametnega telefona. Kot razlago je sodišče zapisalo, da za geslo mora posameznik razkriti nekaj kar ve, prstni odtis pa je fizični dokaz, podobno kot rokopis ali DNK, katere pa oblast lahko (v določenih okoliščinah) zahteva od posameznika.

Pred kratkim je okrožno sodišče druge stopne v Floridi odločilo, da mora obtoženi povedati svoje geslo s katerim je zaklenjen njegov pametni telefon [28]. V razlagi so zapisali, da geslo ni direktno povezano s slikami in video posnetki, ki so ali niso shranjeni na pametnem telefonu.

2.3 Varnostni žetoni

Fraza "nekaj kar uporabnik ima" pomeni, da mora imeti uporabnik pri identifikaciji pri sebi nek konkreten fizičen predmet. Tak primer je ključ od stanovanja. Seveda lahko slednjega izgubimo ali pa naredimo njegovo kopijo in jo damo neki drugi osebi. Tak način overjanja ni popoln, je pa natančen: ključ odklene natanko eno ključavnico in ta ključ odklene samo to ključavnico.[2]

Poznamo več vrst žetonov:

(a) **Pasivni žetoni.** Kot že samo ime pove, gre za žetone, ki se ne spreminjajo. Primer takega žetona je lahko uporabnikova slika ali ključ.

(b) **Aktivni žetoni.** To so žetoni katerih vsebina se ob uporabi spremeni. Primer takega žetona je lahko karta za javni prevoz. Ko jo uporabnik

[2]tukaj namenoma ignoriramo glavni oz. master ključ

prisloni k čitalcu ta prebere njeno trenutno stanje, od njega odšteje ceno potovanja in ga zapiše nazaj na kartico.

(c) **Statični žetoni.** Primer statičnih žetonov so npr. ključi, osebna izkaznica, potni list, kreditne kartice ali druge kartice z magnetnim trakom. Težava teh žetonov (predvsem kreditnih kartic) je, da jih je možno skopirati. Napad se imenuje *skimming*. Pri tem napadu se uporablja posebna naprava, ki prekopira podatke iz magnetnega traku in jih posreduje napadalcu.

(d) **Dinamični žetoni.** Da preprečimo kopiranje statičnih žetonov uporabimo dinamične. Kot že samo ime pove, se vrednost teh žetonov spreminja. Obstaja več različnih načinov, vendar gre v osnovi za napravo, ki generira neko naključno vrednost, katero nato uporabimo kot geslo. Nekatere naprave generirajo vrednosti na nekem časovnem intervalu (npr. vsakih 30 sekund), pri drugih je potrebno pritisniti na gumb in nato generirajo vrednost. Vrednost je lahko generirana tudi kot odgovor na neko vneseno vrednost (izziv). V teh primerih ni pomembno, če nekdo drug vidi ali sliši to vrednost, saj je uporabna samo za eno prijavo. Poleg tega napadalcu ta podatek nič ne pomaga, ker iz njega ne more generirati naslednje vrednosti. Primer takih žetonov sta npr. SecurID iz RSA Laboratories [55] in Google Authenticator. V obeh primerih se žetoni generirajo v nekem časovnem intervalu. Je pa med njima ena bistvena razlika in sicer pri SecureID ima uporabnik tudi svojo PIN kodo katero mora dodati k generirani vrednosti (geslo = PIN koda + žeton), pri aplikaciji Google Authenticator pa je geslo kar sama vrednost žetona.

2.4 Uporaba več faktorjev

Overjanje z enim samim faktorjem ima svoje prednosti in slabosti. Npr. žeton je dober način overjanja, dokler ga uporabnik ne izgubi ali mu ga

ukradejo. Geslo je dobro, dokler ga nekdo ne vidi (npr. ko ga uporabnik vpisuje) in si ga zapomni. Te slabosti lahko (vsaj deloma) odstranimo, če združimo več načinov overjanja. Zelo pogost primer je uporaba dveh faktorjev, tako imenovano dvostopenjsko overjanje. Pri tem običajno uporabljamo geslo in žeton. Sistemi se lahko med seboj nekoliko razlikujejo, običajno predvsem pri načinu "pridobivanja" žetonov. Tako lahko dobimo žeton preko SMS sporočila na mobilni telefon, [3] uporabimo posebno napravo za generiranje žetonov (npr. SecurID) ali pa posebno aplikacijo, ki prav tako generira žetone (npr. Google Authenticator).

Slaba stran overjanja z večjim številom faktorjev je, več dela za uporabnika in s tem možne dodatne nevšečnosti. Tako lahko sistemi nekoliko pretiravajo s številom faktorjev overjanja. Če vzamemo za primer sistem, pri katerem mora uporabnik vpisati štiri različna gesla/PIN kode za dostop. Na prvi pogled bi lahko rekli, da je sistem sigurno bolj varen, kot če bi uporabljali samo eno geslo, saj bi moral napadalec pridobiti štiri gesla. Vendar je lahko to tudi dodatna slabost sistema, saj se bo uporabnik veliko težje zapomnil štiri gesla, kot pa enega samega. Posledično je velika verjetnost, da bo uporabil precej slabša gesla ali pa si jih bo nekam zapisal (npr. na list papirja, katerega bo imel poleg računalnika) in bo s tem precej zmanjšal samo varnost, saj jih bo lahko prebral vsak, ki bo imel dostop do njih.

[3] Pošiljanje žetonov preko SMS sporočil, zaradi ranljivosti mobilnega omrežja, ni več priporočljivo [47].

Poglavje 3

Pregled obstoječih rešitev

Kot je bilo omenjeno že v uvodu, so se za upravljanje gesel pojavile različne aplikacije, ki uporabniku olajšujejo vsakodnevno uporabo gesel. V tem poglavju si bomo podrobneje pogledali nekaj takih aplikacij.

3.1 LastPass

LastPass [40] je ena izmed bolj popularnih aplikacij za hranjenje gesel. Aplikacija se večinoma uporablja preko spletnega vmesnika, na voljo pa so tudi razširitve za večino sodobnih spletnih brskalnikov, poleg tega so na voljo tudi verzije za mobilne telefone. Aplikacija med drugim omogoča:

- **Hranjenje gesel v trezorju.** Vsa gesla in zapiski so varno shranjena v trezorju[1].
- **Avtomatsko vpisovanje gesel.** Ko se uporabnik prijavi/registrira na neko spletno stran, se uporabniško ime in geslo avtomatsko shranita in se ob naslednji prijavi samodejno vpišeta.
- **Več računov.** LastPass omogoča hranjenje več računov za isto spletno stran in enostavno prehajanje med njimi.

[1] Trezor (ang. vault), tako pri LastPassu imenujejo podatkovno bazo v kateri so shranjeni vsi uporabnikovi podatki (npr. uporabniška imena, gesla, itd.)

- **Enostaven dostop.** Trezor z gesli se avtomatsko sinhronizira z vsemi napravami. Prav tako se samodejno ustvarjajo varnostne kopije.
- **Novo geslo za vsak nov račun.** Ob registraciji na spletno stran, aplikacija samodejno predlaga novo naključno geslo in s tem omogoči, da se geslo uporablja samo za eno spletno stran.
- **Analiza gesel.** Z uporabo varnostnega izziva aplikacija omogoča iskanje slabih, starih podvojenih in potencialno ranljivih gesel.
- **Avtomatska menjava gesel.** Aplikacija lahko namesto uporabnika samodejno zamenja geslo.
- **Uporaba prstnega odtisa.** Za prijavo v mobilno aplikacijo LastPass je možno uporabiti prstni odtis uporabnika.
- **Lokalno šifriranje.** Vsi uporabnikovi podatki so šifrirani in dešifrirani na posamezni napravi, tako tudi LastPass nima dostopa do uporabnikovega glavnega gesla in posledično ne more dostopati do podatkov, ki so shranjeni v trezorju.

3.1.1 Varnost

Ko si uporabnik ustvari nov LastPass račun, se na podlagi elektronskega naslova in izbranega gesla (glavno geslo[2]) lokalno generira unikaten ključ, s katerim so šifrirani vsi podatki v uporabnikovem trezorju. Ker šifriranje podatkov poteka lokalno in glavno geslo ter ključi za šifriranje in dešifriranje nikoli ne zapustijo uporabnikove naprave, tudi zaposleni pri LastPass-u ne morejo dostopati do uporabnikovih gesel. Poleg tega je pri prijavi možna uporaba 2-stopenjskega overjanja uporabnika, kar še dodatno poveča varnost [41].

Pri komunikaciji z LastPass strežniki se uporablja SSL, kljub temu, da je večino podatkov že šifriranih s 256-bitnim AES. Na ta način so podatki

[2]Geslo, ki se uporablja za prijavo v aplikacijo LastPass.

neuporabni tako za LastPass, kot tudi za morebitnega napadalca, ki bi na omrežju prestrezal podatke. Politika podjetja je, da nikoli ne prejemajo osebnih podatkov uporabnika, ki niso bili predhodno šifrirani z glavnim geslom, katerega se praviloma nikoli ne pošilja na strežnik. Taka politika močno zmanjša možnosti napadalca. Tako se, kljub uporabi požarnih zidov in najboljših praks za zaščito spletnih servisov in strežnikov, zanašajo predvsem na dejstvo, da če sami ne morejo dostopati do uporabnikovih podatkov, tudi za morebitne napadalce velja enako. Za izdelavo varnega šifrirnega ključa se uporablja večje število krogov PBKDF2-SHA256 (funkcija za izpeljavo ključa iz gesla - ang. Password-Based Key Derivation Function). Vse to precej zmanjša možnost uspešnega napada z grobo silo (vsaj v nekem realnem časovnem obdobju).

Ko se uporabnik prijavi, se na strežnik pošlje samo zgostitev glavnega gesla. Pri zgoščevalni funkciji se za večjo varnost uporablja tudi sol. Le-ta je v tem primeru kar uporabniško ime. Iz definicije zgoščevalne funkcije izhaja, da je iz dobljene zgostitve nemogoče v doglednem času priti nazaj do začetne vrednosti.

Z uporabo soli so dosegli, da napad z obstoječimi mavričnimi tabelami ni mogoč. Za dodatno zaščito so poskrbeli z uporabo PBKDF2-SHA256, kar napadalcu še dodatno oteži delo. Ko vse to združimo skupaj izgleda izračun zgostitve nekako takole (to je zelo poenostavljena predstavitev):

$$zgostitev\ gesla = zgostitev(glavno\ geslo + uporabniško\ ime)^{iteracije}$$

Za izračun zgostitve se na strani klienta naredi 5.000 iteracij [42], kar predstavlja dobro razmerje med hitrostjo in varnostjo. Tako se možnosti za uspešen napada z grobo silo precej zmanjšajo. Z uporabo dobrega gesla se možnosti še dodatno zmanjšajo. Nadalje lahko uporabnik tudi sam poveča število iteracij in s tem še poveča varnost. Na strežniku se zgostitev shrani šele po dodatnih 100.000 iteracijah.

Za šifriranje trezorja se uporablja 256-bitni AES. Njegov ključ se izračuna po že opisanem postopku. Tako se pred pošiljanjem na strežnik vsebina trezorja šifrira, ob prijavi pa se trezor prenese s strežnika in dešifrira.

3.1.2 Varnostni incidenti

V preteklosti so zaznali dva napada na strežnike LastPass. Leta 2011 so na omrežju zaznali anomalije [44]. Analiza ni pokazala nobenih znakov klasičnega vdora (npr. da bi kakšen naveden uporabnik postal administrator), prav tako pa niso odkrili vzroka anomalij. Glede na promet na omrežju je bilo teoretično možno, da je prišlo do kraje elektronskih naslovov, zgostitev gesel ter soli, ki se uporablja za izračun zgostitev na strežniku. Da bi zmanjšali tveganje, so odstranili vse strežnike, ki so bili kompromitirani. O dogajanju so sproti obveščali uporabnike in so jih prosili, da zamenjajo glavno geslo. Slednje je povzročilo dodatne težave, ker so bili strežniki preobremenjeni. Kot so kasneje sporočili, ni bilo nobenih potrjenih poročil o izgubljenih uporabnikovih podatkih ali geslih. Za povečanje varnosti podatkov so na strežnikih vpeljali uporabo PBKDF2-SHA256 s 256-bitno soljo in 100.000 iteracijami.

Do podobnega napada je prišlo tudi leta 2015, ko so na omrežju odkrili sumljiv promet [45]. Preiskava dogodkov ni odkrila nobenih dokazov, da je prišlo do kraje uporabniških trezorjev. Vendar se je izkazalo, da so bili elektronski naslovi, opomniki gesel, zgostitev gesel ter soli, ki se uporablja za izračun zgostitev na strežniku, kompromitirani. Bili so prepričani, da so njihovi algoritmi dovolj dobri, da napadalcem ukradeni podatki ne bodo nič koristili, so pa uporabnikom svetovali, da za vsak slučaj, zamenjajo glavno geslo. Po tem napadu so še povečali varnost z uporabo HSM-jev (Hardware Security Modules), v katerih so sedaj shranjene zgostitve gesel.

V začetku leta je na hekerski konvenciji, varnostni raziskovalec Sean Cassidy predstavil možen napad na LastPass aplikacijo z ribarjenjem [43]. Pri napadu je uporabnik preusmerjen na zlonamerno spletno stran, ki generira obvestilo podobno LastPass aplikaciji. Lažno obvestilo pogosto pretenta uporabnika, saj slednji misli, da je obvestilo pravo in da ni prijavljen v aplikacijo LastPass, nato pa ga preusmeri na prijavno stran, kjer vnese svoje podatke. Na tak način lahko napadalec pridobi uporabnikovo uporabniško ime in geslo. Z njim se nato prijavi v aplikacijo na svojem računalniku in tako pridobi do-

stop do vseh njegovih gesel. Čeprav slednje ni neposredna slabost aplikacije, so naredili nekaj izboljšav, ki zmanjšujejo tveganje takega napada. Sedaj lahko uporabnik preko razširitve LastPass, ki je v orodni vrstici brskalnika, vidi, ali je prijavljen v aplikacijo. Če dobi obvestilo, da se mora prijaviti, ve, da ga želi napadalec z obvestilom zavesti. Če uporabnik spregleda, da je že prijavljen in se želi prijaviti, aplikacija samodejno zazna, da uporabnik vpisuje glavno geslo v obrazec, ki ni od LastPass-a in ga o tem obvesti. Možno pa je, da zlonamerna spletna stran prepreči oz. skrije vsa LastPass opozorila. V tem primeru uporabnik ni opozorjen in vseeno vpiše svoje podatke. Tako napadalec pride do vseh podatkov, katere rabi za prijavo. Ko pa se napadalec poskuša prijaviti v aplikacijo, ta zazna, da se uporabnik prijavlja z novo napravo oz. iz nove lokacije, zato aplikacija za prijavo zahteva dodatno potrditev po elektronski pošti. Slednje močno zmanjša tveganje za uspešen napad z ribarjenjem, saj bi napadalec moral prej pridobiti tudi dostop do elektronske pošte uporabnika.

Kot je razvidno iz preteklih napadov, so vedno sproti obveščali svoje uporabnike ter jim svetovali, kako naj ukrepajo. Poleg tega so še dodatno povečali varnost in s tem zmanjšali tveganje za morebiten uspešen napad. Vse to še dodatno povečuje zaupanje v aplikacijo LastPass.

3.2 KeePas

KeePass [36] je namenska, odprtokodna aplikacija za varno hranjenje gesel. Vsi podatki so shranjeni v podatkovni bazi, ki je šifrirana z glavnim geslom oziroma z datoteko.[3]

Aplikacija omogoča med drugim naslednje funkcionalnosti:

- **Avtomatsko izpolnjevanje obrazcev.** Če je v ozadju aplikacija odprta, lahko z globalno bližnjico poišče ustrezne podatke in jih vnese v izbrano vnosno polje.

[3]geslo je lahko shranjeno v datoteki in ker si ga uporabnik ne potrebuje zapomniti je lahko le-to daljše

- **Varno upravljanje z odložiščem.** Z dvoklikom na katerokoli shranjeno vrednost se ta prekopira v odložišče, ki pa se čez nekaj časa samodejno izbriše.

- **Iskanje in razvrščanje.** Po podatkovni bazi je možno iskati specifičen podatek. Podatke v posamezni skupini je možno razvrščati po poljubnem stolpcu.

- **Podpora za več jezikov.** Aplikacijo je možno prevesti v več jezikov. Trenutno je podprtih več kot 30 jezikov.

- **Generator naključnih gesel.** V aplikaciji je vgrajen generator naključnih gesel. Uporabnik lahko določi število in tip znakov.

- **Odprtokodni program.** Končni uporabnik ima popoln dostop do izvorne kode. Tako jo lahko vsak posameznik pregleda in preveri, če so algoritmi pravilno implementirani. To uporabniku omogoča, da si lahko sam izbere tudi kakšen drug kriptografski algoritem.

- **Enostaven prenos podatkovne baze.** Celotna podatkovna baza se nahaja v eni datoteki, katero lahko enostavno prenašamo iz enega računalnika na drugega. Za potrebe avtomatskega sinhroniziranja podatkov med več napravami je možno podatkovno bazo shraniti v oblak (npr. Dropbox).

- **Prenosna, uporaba brez namestitve.** Aplikacijo lahko prenesemo z USB ključem in jo uporabljamo v tem primeru le v MS Windows sistemu brez predhodne namestitve.

3.2.1 Varnost

Za šifriranje podatkov v podatkovni bazi sta na voljo dva algoritma:

(a) AES (velikost bloka - 128 bitov; velikost ključa - 256 bitov) v CBC načinu.

(b) Twofish (velikost bloka - 128 bitov; velikost ključa - 256 bitov)

Oba veljata za varna algoritma, saj sta bila v ožjem izboru za nov standard (AES).

Šifrirani so vsi shranjeni podatki, kot so npr. zapiski, uporabniška imena, spletni naslovi, itd. [37], in ne samo gesla. Geslo, ki se uporablja za šifriranje podatkov v podatkovni bazi, se generira iz glavnega gesla z zgoščevalno funkcijo SHA-256. Z drugimi besedami, za šifriranje se uporablja zgostitev glavnega gesla. Pri tem načinu se pri izračunu zgostitve doda tudi 128-bitno sol, kar preprečuje morebitne napade z mavričnimi tabelami.

Ker je varnost podatkov zelo odvisna od "kvalitete" glavnega gesla, aplikacija omogoča tudi alternativo. Tako je ključ lahko shranjen v datoteki, kar v večini primerov zagotavlja boljšo varnost, saj je lahko občutno daljši. Datoteko lahko shranimo na npr. USB ključu, zgoščenki, itd. Seveda pa mora biti uporabnik pazljiv, da ga ne izgubi. V tem primeru je priporočljivo imeti, na nekem varnem mestu, varnostno kopijo ključa.

Za še večjo varnost pa je možna tudi kombinacija obeh metod. Tako je potrebno za dostop do baze podatkov, ključ iz datoteke ter glavno geslo. V tem primeru so podatki varni tudi, če uporabnik izgubi datoteko. Šifrirno geslo se izračuna na način:

```
geslo = SHA-256(SHA-256(glavno geslo), ključ iz datoteke)
```

V primeru, ko je ključ v datoteki krajši od 256-bitov, se pred združitvijo izračuna tudi zgostitev ključa:

```
geslo = SHA-256(SHA-256(glavno geslo), SHA-256(ključ iz datoteke))
```

Poleg tega je na sistemih Windows možno podatkovno bazo povezati s uporabnikovim računom. Tako lahko do podatkov dostopa samo oseba, ki jo je kreirala.

Za generiranje šifrirnega gesla, se najprej izračuna zgostitev glavnega gesla z zgoščevalno funkcijo SHA-256. Rezultat se nato N-krat šifrira z algoritmom AES, iz katerega se potem ponovno izračuna zgostitev (SHA-256).

Pri šifriranju z algoritmom AES se uporablja naključni 256-bitni ključ, ki je shranjen v podatkovni bazi. Ker ni možno v naprej izračunati rezultata AES, mora napadalec za vsak ključ posebej izračunati celoten postopek. To napadalca precej upočasni in s tem zmanjša možnost njegovega uspeha. Privzeta vrednost za N je 6000. Vrednost je prilagojena tudi mobilnim napravam, katere imajo manj zmogljive procesorje. Če uporabnik namerava uporabljati podatkovno bazo samo na računalniku je priporočljivo, da vrednost N poveča.

Taka zaščita je uporabljena samo v primeru glavnega gesla. Gesla v datoteki so v osnovi naključne vrednosti, tako da ni potrebe za enak postopek. Uganiti tako geslo je enako zahtevno kot odkriti geslo z uporabo napada z grobo silo.

V času, ko je aplikacija aktivna, so v pomnilniku shranjeni občutljivi podatki, kot je npr. šifrirno geslo. Ti podatki so šifrirani, tako da če bi kdo shranil podatke iz pomnilnika na disk, si z njimi ne bi mogel veliko pomagati. Za dodatno zaščito, pa se ti podatki prepišejo z naključnimi vrednostmi, preden aplikacija sprosti lokacije v pomnilniku.

Vsakič, ko se aplikacija zažene, se sproži hitro samodejno testiranje. Pri tem se preveri, če vsi šifrirni algoritmi in zgoščevalne funkcije delujejo pravilno. Če pri katerem od testov pride do napake, se o tem obvesti uporabnika.

3.2.2 Varnostni incidenti

V začetku leta 2016 je prišlo v javnost, da ima postopek preverjanja in prenosa posodobitev, varnostno luknjo [49]. V fazi preverjanja posodobitve promet ni bil šifriran, kar je napadalec lahko izkoristil. V tem primeru je napadalec lahko izvedel tako imenovani napad z možem v sredini (ang. man in the middle) in namesto, da bi uporabnik dobil novo verzijo aplikacije, je dobil zlonamerno programsko kodo (ang. malware). Po nekaterih podatkih, je bila to precej znana ranljivost, katere so se programerji zavedali, vendar je niso želeli odpraviti, ker bi potem imeli težave s prikazovanjem oglasov in posledično manjših prihodkov.

V odgovor na obtožbe so pri KeePassu dejali, da aplikacija omogoča samo preverjanje za posodobitve in ne omogoča avtomatskega prenosa in namestitve aplikacije [38]. Poleg tega bi moral po prenosu posodobitve iz spleta, vsak uporabnik preveriti digitalni podpis aplikacije in se na ta način prepričati, da je prava aplikacija. Za rešitev nastale situacije so poskrbeli, da je datoteka s številko verzije nove aplikacije digitalno podpisana in po novem, aplikacija sprejme samo datoteke, ki so digitalno podpisane. Poleg tega sedaj komunikacija poteka preko HTTPS povezave.

3.3 iCloud Keychain

iCloud Keychain je servis, ki omogoča:

- generiranje naključnih gesel,
- sinhronizacija podatkov (gesel, podatkov kreditnih karticah, gesla brezžičnih omrežij, itd.) med vsemi uporabnikovimi napravami, ki uporabljajo operacijska sistema iOS ali macOS,
- izdelava varnostne kopije, ki je shranjena na Applovih strežnikih

Pri načrtovanju iCloud Keychaina so si zadali cilj, da so uporabnikova gesla varna tudi v primeru ko:

- je uporabnikov iCloud račun kompromitiran,
- je iCloud kompromitiran s strani zunanjega napadalca ali zaposlenega pri Applu,
- ima neka tretja oseba dostop do uporabnikovega računa.

3.3.1 Varno sinhroniziranje podatkov

Ko uporabnik prvič vključi iCloud Keychain, naprava vzpostavi krog zaupanja in kreira svojo sinhronizacijsko identiteto, ki jo sestavlja par javnega in

zasebnega ključa. Javni ključ je šifriran z zasebnim ključem nato še s ključem, ki je izpeljan iz uporabnikovega iCloud gesla. Ta podatek se nato shrani v iCloud (zasebni ključ nikoli ne zapusti naprave na kateri je bil kreiran). Tako je na iCloudu shranjen javni ključ naprave, ki je šifriran z uporabnikovim geslom za iCloud ter digitalno podpisan z zasebnim ključem naprave.

Ko uporabnik doda novo napravo, se enak postopek ponovi tudi na tej napravi. Šifriran in digitalno podpisan javni ključ nove naprave se shrani v iCloud ter na vse uporabnikove naprave, ki uporabljajo iCloud Keychain.

Ob dodajanju oz. spremembi gesla, se z ostalimi napravami, sinhronizira samo posamezen keychain element (eden po eden). Z drugimi besedami, vsak keychain element je poslan samo na napravo, katera rabi posodobitev. Vsak tak element je šifriran tako, da ga lahko dešifrira samo vsaka posamezna naprava. Tako bi moral zaposleni pri Applu kompromitirati osnovno iCloud arkitekturo na več mestih, da bi lahko prišel do gesel uporabnikov. Niti kraja uporabnikovega iCloud uporabniškega imena in gesla, napadalcu ne bi omogočila dostopa do gesel. Napadalec bi moral imeti tudi dostop do uporabnikove naprave, ki uporablja iCloud Keychani, da bi odobril novo (svojo) napravo in tako prišel do uporabnikovih gesel.

3.3.2 Varna obnova podatkov

Za razliko od sinhroniziranja podatkov, kjer se hkrati pošilja samo en keychain element, iCloud Keychain Recovery shrani varnostno kopijo vseh keychain podatkov. Celotna varnostna kopija je šifrirana s sejnim ključem in shranjena v iCloud. Nato se varnostna kopija dešifrira in ponovno šifrira z novim ključem (iCloud Security Code) in javnim ključem namenske strojne opreme poznane kot HSM (ang. Hardware Security Module). Bolj natančen opis postopka je dostopen v [14].

Obnovitveni postopek zahteva tudi uporabnikovo telefonsko številko, saj mora uporabnik, v primeru obnove, odgovoriti na poslano sporočilo. Poleg tega mora uporabnik vnesti svoje iCloud uporabniško ime in geslo ter iCloud Security Code za dešifriranje podatkov. V primeru, da pri postopku obnove

pride do 10-ih napak, HSM izbriše ključ tega uporabnika, kar pomeni, da je dostop do podatkov varnostne kopije popolnoma onemogočen.

Za vsak slučaj, so pri Applu uničili administratorske dostopne kartice za HSMje in jih nastaviti tako, da v primeru zaznave nepooblaščenega dostopa, izbrišejo vse ključe. Nato sistem obvesti vse uporabnike, ki so bili vezani na ta HSM, da se morajo ponovno vpisati. Pri ponovnem vpisu se uporabi drug HSM.

3.3.3 Varnostni incidenti

Skupini šestih univerzitetnih raziskovalcev je uspelo narediti aplikacijo, ki je dostopala do keycahin podatkov drugih aplikacij [60]. V osnovi se aplikacija nahaja znotraj peskovnika in ima dostop samo do svojih podatkov, ne pa tudi do podatkov drugih aplikacij. Tukaj ne gre za ranljivost iCloud Keychaina, ampak za ranljivost sistema komunikacije med aplikacijami znotraj operacijskega sistema. Poleg tega jim je uspelo tako aplikacijo tudi objaviti na App Storu, kar je razkrilo pomanjkljivosti pri Applovem pregledu aplikacij.

3.4 Projekt Abacus

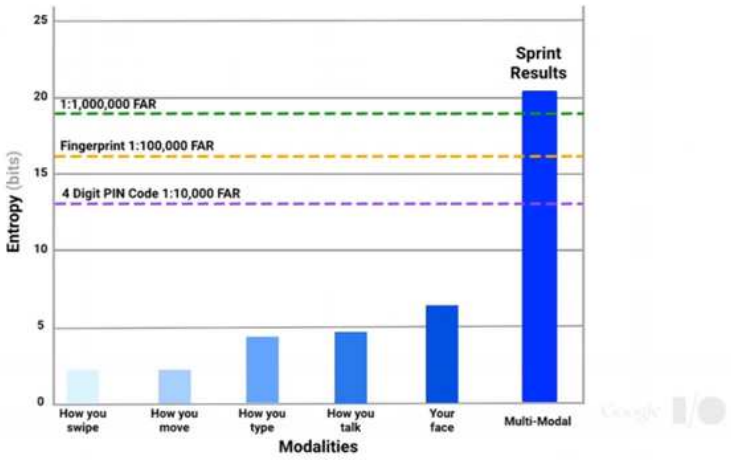
Projekt Abacus je bil predstavljen na Googlovi I/O konferenci leta 2015 [30]. Pri predstavitvi so podarili, da si ljudje težko zapomnimo gesla, poleg tega so običajna gesla relativno slaba. Po podatkih Googla 70% uporabnikov, enkrat na mesec, pozabi svoje geslo in v povprečju vpišejo 2,4 gesel preden vpišejo pravo geslo [13].

Za rešitev te težave predlagajo uporabo biometričnih podatkov. Nove naprave so običajno opremljene s čitalcem prstnih odtisov, samo veliko uporabnikov še vedno uporablja starejše naprave, brez ustrezne strojne opreme. Zaradi tega so predlagali uporabo podatkov iz drugih senzorjev, ki omogočajo prepoznavo:

- vzorca tipkanja,

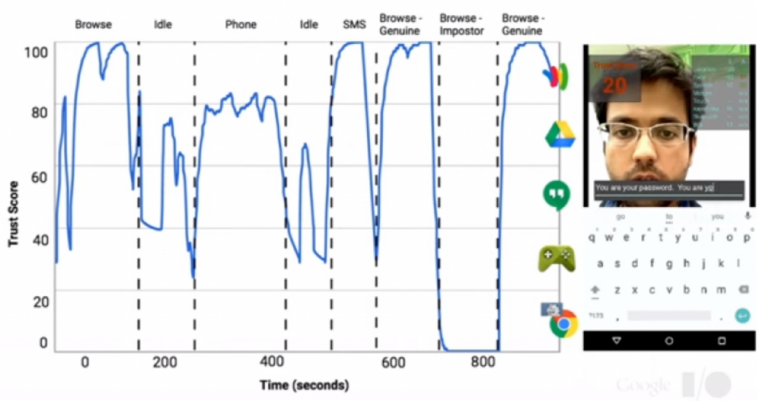
- vzorca hoje,
- vzorca govora,
- obraza
- itd.

Ker ti vzorci ne zagotavljajo dovolj velike entropije in posledično tudi ne dovolj zanesljivega overjanja, so postavili hipotezo, da se bosta tako entropija kot tudi stopnja zanesljivosti povečala, če vse te podatke združijo. Za dokaz te hipoteze so najprej zbrali podatke 1500 prostovoljcev, katere so nato analizirali. Rezultat analize je bila metoda, ki omogoča 10-krat bolj natančno overjanje kot prstni odtis in 100-krat varnejše kot 4 mestna PIN koda [31]. Primerjava entropije je prikazana na Sliki 3.1.



Slika 3.1: Primerjava entropije posameznih načinov overjanja.

Metoda se stalno izvaja v ozadju delovanja naprave in neprestano izračunava oceno zaupanja. Tako lahko aplikacije v vsakem trenutku dostopajo do ocene zaupanja in se temu primerno odzivajo. Različne aplikacije imajo različne zahteve, tako npr. neka igra ne potrebuje visoke stopnje zaupanja, na drugi strani pa neka bančna aplikacija zahteva visoko stopnjo zaupanja. Primer



Slika 3.2: Prikaz sprotnega izračunavanja ocene stopnje zaupanja.

stalnega izračunavanja stopnje zaupanja in zahtevane stopnje zaupanja posamezne aplikacije je dobro viden na Sliki 3.2.

Na tem mestu se pojavi vprašanje uporabnikove zasebnosti in kam se shranjujejo vsi tako zbrani podatki. Po podatkih Googla se vsi izračuni izvajajo lokalno na posamezni napravi [31], kljub temu pa se morajo shranjevati na lokalni pomnilnik. To bi lahko pripeljalo do raznih zlorab podatkov, saj bi lahko nepooblaščene aplikacije dostopale do njih. Dodatno vprašanje je, kaj se zgodi s temi podatki, ko uporabnik kupi nov pametni telefon. Če se ti podatki prenesejo preko varnostne kopije v oblaku, to pomeni, da ima Google dostop do njih.

Če za primerjavo vzamemo Applov Touch ID, ki uporablja prstni odtis za potrebe overjanja uporabnika, ga sestavlja senzor za prepoznavo prstnih odtisov ter poseben čip, ki je namenjen hranjenju teh podatkov. To je poseben čip do katerega aplikacije nimajo dostopa. Poleg tega se ti podatki nikoli ne shranijo v varnostno kopijo, tako da se tudi ne prenesejo v iCloud in posledično mora uporabnik na vsaki (novi) napravi posebej vnesti podatke o svojih prstnih odtisih.

Poglavje 4

Uporabljene tehnologije

Za osnove kriptografije priporočamo Stinsonov učbenik za Kriptografijo - teorija in praksa [8], vse podrobnosti o kriptografskih algoritmih pa najdemo v obsežnem Priročniku za Uporabno kriptografijo, glej Menezes et al. [3].

4.1 AES

AES je kratica za Advanced Encryption Standard in je naslednik simetrične bločne šifre DES (Data Encryption Standard) [9, Sec. 2.12]. DES sta razvila Nacionalni inštitut za standarde in tehnologijo (NIST) ter IBM in je bil leta 1976 sprejet kot standardni kriptografski algoritem v ZDA [23]. Do leta 1997 je bila dovoljena uporaba algoritma samo znotraj ZDA, izvoz pa je bil prepovedan.

Ker so se pojavili pomisleki glede fiksne dolžine šifrirnega ključa (56-bitov), ki ga je uporabljal DES in dejstva, da se računska moč računalnikov stalno povečuje, so varnostni analitiki pričeli z iskanjem zamenjave za DES. Tako je januarja leta 1997 NIST objavil natečaj za novo simetrično bločno šifro. Med pomembnejšimi zahtevami je bila, da mora biti algoritem javno dostopen vsakomur brez licenčnih plačil. Uporaba algoritma ne sme biti omejena na posamezne države ali s patentnimi zahtevki. Ena od zahtev je bila tudi, da mora biti simetrična bločna šifra, ki deluje na blokih dolžine

vsaj 128 bitov. Da se ne bi pojavila enaka težava s ključi kot pri DESu, je bila postavljena zahteva za ključe dolžin 128, 192 in 256 bitov. Podrobno poročilo o razpisu je možno najti v [52].

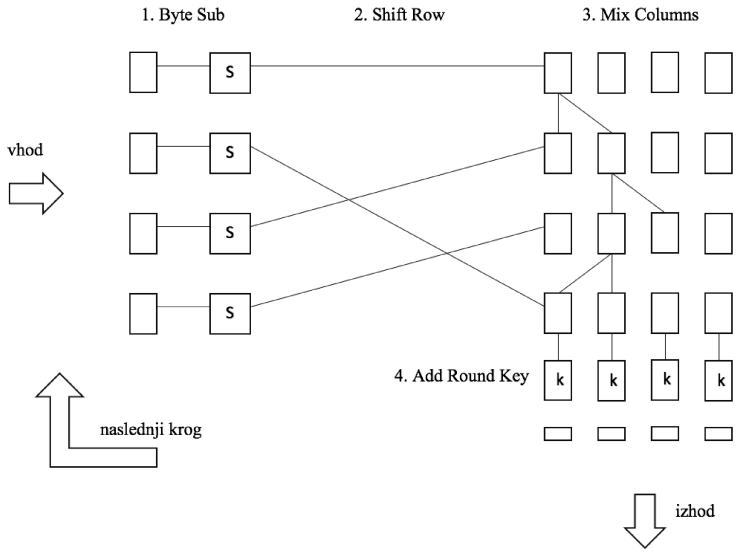
Izbor algoritma je potekal v treh fazah. V prvi fazi, avgusta 1998, je bilo med predlaganimi algoritmi izbranih 15, ki so zadostili vsem razpisnim pogojem. V drugi fazi, ki se je zaključila avgusta 1999, so izbrali naslednjih 5 finalistov:

- MARS,
- RC6,
- Rijndael,
- Serpent in
- Twofish.

Teh pet algoritmov je bilo deležnih obsežnih javnih in zasebnih varnostnih testiranj. Zadnji izbor je temeljil na varnosti, stroških oziroma učinkovitosti in enostavnosti implementacije v programsko opremo. Med temi petimi algoritmi ni bilo odkritih nobenih kriptografskih napak, tako je bil končni izbor narejen glede na učinkovitost in karakteristike posameznih implementacij. Kot zmagovalni algoritem je bil izbran Rijndael.

AES podpira velikost blokov 128, 192 in 256 bitov ter velikost ključev 128, 192 in 256 bitov, ki ju je možno izbrati neodvisno. Zasnovan je bil z namenom, da bi učinkovito deloval na 8-bitnih (pametne kartice) in 32-bitnih (osebni računalniki) procesorjih. V tem pogledu je Rijndael najbolj fleksibilen med vsemi finalisti.

Vsi izračuni potekajo v končnem obsegu GF(2^8). Vsak element v tem obsegu je možno zapisati z enim bajtom informacije, kar je ugodno za implementacijo. Za predstavitev obsega GF(2^8) v Rindaelu, uporabljamo polinomsko bazo z nerazcepnim polinomom $x^8 + x^4 + x^3 + x + 1$. Algoritem



Slika 4.1: Struktura AES.

na več mestih operira tudi s polinomi nad obsegom GF(2^8), predvsem s polinomi stopnje kvečjemu 3. Vsak tak polinom nosi 32 bitov informacije, kar je posebej ugodno za 32-bitne procesorje.

	$B = 128$	$B = 192$	$B = 256$
$K = 128$	10	12	14
$K = 192$	12	12	14
$K = 256$	14	14	14

Tabela 4.1: Število krogov šifriranja v odvisnosti od velikosti bloka (B) in velikosti ključa (K).

Z razliko od predhodnika (DES), AES ne uporablja standardne Feistelove strukture, vseeno pa šifriranje poteka v več zaporednih krogih. Število krogov je odvisno od velikosti ključa in velikosti bloka, kar je prikazano v Tabeli 4.1.

Vsak krog je sestavljen iz štirih bijektivnih transformacij (Slika 4.1):

- SubBytes,

- ShiftRows,
- MixColumns,
- AddRoundKey.

Vsak krog šifriranja, z izjemo zadnjega, ima enako strukturo. Pri zadnjem izpustimo transformacijo MixColumns. To služi enotnosti zgradbe šifriranja in dešifriranja. Pred prvim krogom šifriranja podatkom bitno prištejemo prvi pod-ključ. V nasprotnem primeru bi napadalec lahko transformacije iz prvega kroga enostavno odstranil, saj le-te niso odvisne od ključa. Za podrobnosti Rijndaelovega algiritma priporočamo Landau [5], (kjer so opisani tudi drugi finalisti AES izbora bločne simetrične šifre za 21. stoletje), cf. [11], za varnostno študijo pa Courtois in Pieprzyk [1].

4.2 Zgoščevalne funkcije

Zgoščevalna funkcija je katerakoli funkcija, ki podatke poljubne dolžine preslika v podatke fiksne dolžine [33]. Vhodne podatke v funkcijo običajno imenujemo "sporočilo" izhod pa "zgostitev" (povzetek, prstni odtis). Kriptografske zgoščevalne funkcije so posebna skupina zgoščevalnih funkcij, ki ustrezajo določenim kriterijem, kar jih naredi primerne za uporabo v kriptografiji [20]. Od kriptografske zgoščevalne funkcije h pričakujemo naslednje lastnosti [3, Ch. 9]:

- **odpornost na prasliko** (ang. pre-image resistance). Pri dani zgostitvi H v doglednem času ni možno najti sporočila M', tako da velja

$$h(M') = H.$$

- **Odpornost na 2. prasliko** (ang. second pre-image resistence). Pri danem sporočilu M in njegovi zgostitvi $H = h(M)$, v doglednem času ni možno najti drugega sporočila M', tj. $M' \neq M$, z enako zgostitvijo H.

- **odpornost na trke** (ang. Collision resistance)[1]. V doglednem času ni možno najti dveh različnih sporočil M in M', tj. $M' \neq M$, z enako zgostitvijo H.

Zgoščevalna funkcija, ki zadosti prvima dvema pogojema, je enosmerna, kar pa je pogoj za tretjo lastnost. Poleg že naštetih lastnosti morajo dobre zgoščevalne funkcije zadostiti tudi drugim lastnostim. Omenimo samo dve (več v [3, Sect. 9.2]):

- **lastnost plazu** – sprememba enega bita vhoda spremeni približno polovico bitov izhoda,

- **lokalna enosmernost** – pri dani zgostitvi je enako težko dobiti del vhodnega sporočila, kot celotno vhodno sporočilo.

Enosmerne zgoščevalne funkcije se uporabljajo za več namenov [9, Sect. 2.4]:

- za digitalne podpise (skupaj z algoritmi za kriptografijo z javnim ključem),

- overjanje,

- preverjanje integritete,

- v komunikacijskih protokolih.

Zgoščevalne funkcije so najbolj pogosto uporabljeni kriptografski primitivi. Lahko bi rekli, da predstavljajo osnovne gradnike današnje kriptografije [10]. Bruce Schneier je označil enosmerne zgoščevalne funkcije za "delovne konje moderne kriptografije" [18].

[1]Vsaka zgoščevalna funkcija bo imela trke, ker funkcija iz relativno velikega sporočila generira relativno majno zgostitev. Nemogoče je iz 512-bitnega sporočila generirati 128-bitno zgostitev brez trkov. Tukaj je najbolj pomembno, da so trki nepredvidljivi. Vemo, da bo prišlo do trka, vendar je nemogoče predvideti, kateri dve sporočili bosta generirali isto zgostitev.

4.2.1 SHA-1

Najbolj pogosto uporabljene kriptografske zgoščevalne funkcije so MD4, MD5 in SHA (Secure Hash Algorithm). Algoritma MD4 in MD5 je, v letih 1990 in 1992, razvil Ronald Rivest. MD5 je izboljšana verzija algoritma MD4. Oba algoritma iz sporočila generirata 128-bitno zgostitev.

Leta 1993 je NIST objavil SHA-0, ki je temeljila na MD4/MD5 in jo je načrtovala NSA. Naslednje leto je NIST objavil, da v algoritmu SHA-0 obstaja tehnična pomanjkljivost (podrobnosti niso razkrili), zaradi česar je manj varen. Posledica tega je bil nov algoritem SHA-1 (standard FIPS 180-1), ki je bil objavljen leta 1995 [56].

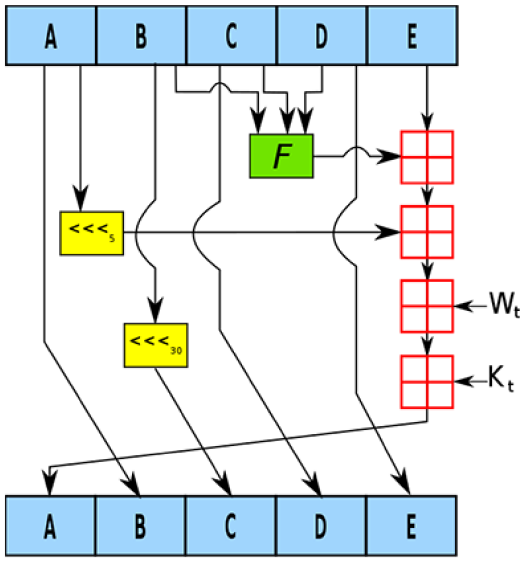
SHA-1 je kriptografska zgoščevalna funkcija, ki jo je razvila Ameriška agencija za nacionalno varnost (NSA) v okviru zveznega standarda (Federal Information Processing Standard – FIPS), ki ga je objavil NIST. Velikost vhodnega sporočila je omejena na 2^{64} bitov. Za generiranje zgostitve je potrebno 80 iteracij krožne funkcije (v primerjavi z MD5, ki potrebuje 64 iteracij). Prikaz ene iteracije je prikazan na Sliki 4.2. Kot rezultat funkcija vrne zgostitev velikosti 160 bitov. Za podroben opis zgradbe in delovanja funkcije glej Stinson [8, Sect. 4.3.2], cf. [53].

Zgoščevalna funkcija SHA-1 ne velja več za dovolj varno (vsaj v primeru napadalca z dovolj sredstvi npr. obveščevalne agencije). Leta 2005 so raziskovalci odkrili napad, ki za iskanje trkov potrebuje manj kot 2^{69} operacij [19]. Od leta 2010 veliko organizacij priporoča zamenjavo SHA-1 s SHA-2 ali SHA-3.

4.2.2 SHA-2

Leta 2008 je NIST objavil nov standard za zgoščevalne funkcije FIPS 180-3 [27]. Standard definira algoritme, ki temeljijo na SHA algoritmu, vendar vračajo večje zgostitve (od 224 do 512 bitov). Ti algoritmi so znani pod oznako SHA-2 [9, Sect. 12.4].

SHA-2 vsebuje znatne razlike v primerjavi z njenim predhodnikom (SHA-



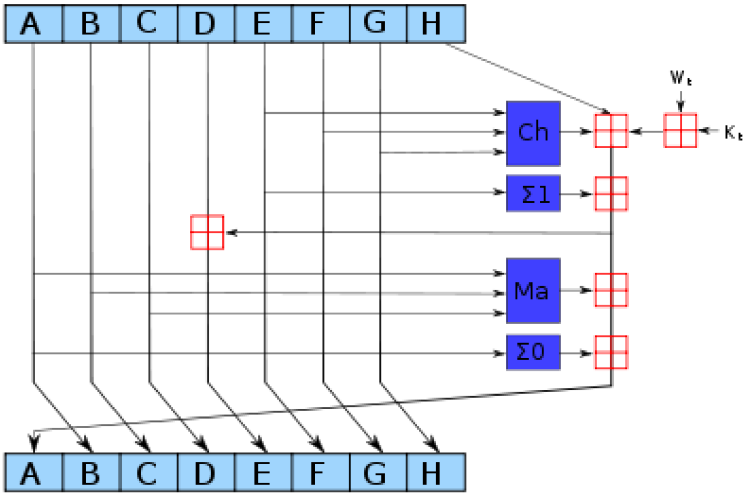
Slika 4.2: Ena iteracija znotraj SHA-1 zgoščevalne funkcije.

1). Standard SHA-2 je sestavljen iz šestih zgoščevalnih funkcij: SHA-224, SHA256, SHA-384, SHA-512, SHA-512/224 in SHA-512/256. SHA-256 in SHA-512 sta novi zgoščevalni funkciji, ki za izračune uporabljata 32 oz. 64-bitne besede. Velikost vhodnega sporočila je omejena na 2^{64} bitov oz. na 2^{128} bitov. Za generiranje zgostitve je potrebno 64 oz. 80 iteracij (odvisno od izbrane funkcije) [57]. Bolj podrobna razdelitev je prikazana v Tabeli 4.3. Predstavitev ene iteracije je prikazana na Sliki 4.3. Podrobnejši opis zgradbe in delovanja funkcij je dosegljiv v [27].

4.2.3 SHA-3

Leta 2008 je NIST objavil natečaj za izbor novega zgoščevalnega algoritma poznanega pod oznako SHA-3. Tako je leta 2012 NIST javno oznanil, da je algoritem Keccak [58] zmagovalec. Tako kot SHA-2 je tudi SHA-3 sestavljen iz več zgoščevalnih funkcij [9, Sect. 12.4]. Standard SHA-3 je bil javno objavljen leta 2015 [48].

Za razliko od predhodnikov je struktura SHA-3 precej drugačna. Tako



Slika 4.3: Ena iteracija znotraj SHA-2 zgoščevalne funkcije.

potrebuje precej manj iteracij (24), kar pripomore k hitrejšemu izvajanju. Potrebuje pa več pomnilnika (1600 bitov v primerjavi s SHA-2, ki potrebuje 512 bitov). Primerjava lastnosti posameznih algoritmov je prikazana v Tabeli 4.3. Podrobnejši opis zgradbe in delovanja funkcij je dosegljiv v [26].

4.3 Funkcije za izpeljavo ključev

Funkcije za izpeljavo ključa (ang. Key Derivation Functions – KDF), na podlagi osnovnega ključa in nekaterih drugih parametrov, izpeljejo nov ključ. V primeru funkcije, ki iz gesla izpelje nov ključ (ang. password-based key derivation function – PBKDF), se za osnovni ključ uporabi podano geslo, za dodatne parametre pa se uporablja sol in število ciklov [54]. KDF se lahko uporablja v primerih, ko je potrebno ključe pretvoriti v daljše ključe oz. v ključe ustreznega formata.

V nasprotju z ostalimi funkcijami, ker je željeno, da je implementacija čim bolj optimizirana in da se izračuni izračunavajo čim hitreje, je v tem primeru želja, da so izračuni čim počasnejši. Razlog za to je predvsem v

Algoritem	Max. dolžina sporočila (bit)	Velikost bloka (bit)	Št. iteracij	Velikost zgostitve (bit)
MD5	2^{64}	512	64	128
SHA-1	2^{64}	512	80	160
SHA-2-224	2^{64}	512	64	224
SHA-2-256	2^{64}	512	64	256
SHA-2-384	2^{128}	1024	80	384
SHA-2-512	2^{128}	1024	80	512
SHA-3-256	∞	1088	24	256
SHA-3-512	∞	576	24	512

Tabela 4.2: Lastnosti zgoščevalnih funkcij.

zaščiti pred morebitnimi napadi na začetno geslo z grobo silo ali s slovarjem, saj bi napadalec, za uspešen napad, rabil preveč časa. Poleg tega se uporablja tudi sol, ki ščiti pred napadi z mavrično tabelo [39].

4.3.1 PBKDF1

PBKDF1 za funkcijo izpeljave ključa uporablja eno od zgoščevalnih funkcij: MD2, MD5 ali SHA-1. Dolžina izpeljanega ključa je omejena z dolžino zgostitve dobljene iz zgoščevalne funkcije (maksimalno 160 bitov). Uporaba PBKDF1 je priporočljiva samo v primerih, ko zaradi združljivosti z obstoječimi aplikacijami ni možna uporaba drugih funkcij. Težava PBKDF1 je predvsem v dolžini izpeljanih ključev, saj za nekatere aplikacije niso dovolj dolgi. Podrobnejši opis delovanja je dosegljiv v [54].

4.3.2 PBKDF2

PBKDF2 zamenjuje PBKDF1 in za izpeljavo ključa uporablja psevdo-naključno funkcijo, kot je npr. zgoščevalna funkcija. Dolžina izpeljanega ključa je v osnovi neomejena, čeprav jo omejuje izbrana psevdo naključna funkcija. Podrobnejši opis delovanja je dosegljiv v [54].

Simetrične šifre (AES)	Asimetrične (RSA, DSA, DH)	Eliptične krivulje
40 bitov	274 bitov	80 bitov
56 bitov	384 bitov	112 bitov
64 bitov	512 bitov	128 bitov
80 bitov	1.024 bitov	160 bitov
96 bitov	1.536 bitov	192 bitov
112 bitov	2.048 bitov	224 bitov
120 bitov	2.560 bitov	240 bitov
128 bitov	3.072 bitov	256 bitov
256 bitov	15.360 bitov	512 bitov

Tabela 4.3: Primerjava dolžine ključev posameznih algoritmov za primerljivo stopnjo varnosti.

Ko je bil leta 2000 standard napisan, je bilo priporočeno, da je najmanjše število ciklov 1000, vendar naj bi se to število večalo s časom (in razvojem hitrejših procesorjev). Od leta 2005, standard Kerberos priporoča uporabo 4096 ciklov [39].

4.4 Eliptične krivulje

Kriptosistemi z eliptičnimi krivuljami (ang. Elliptic Curve Cryptosystem - ECC) sta predlagala Victor Miller in Neil Koblitz leta 1985, kot alternativni algoritem za implementacijo kriptografije z javnim ključem. Za razliko od RSA, ECC temelji na težavnosti računanja logaritmov v končnih obsegih. Prednost ECC je predvsem v dolžini ključev, saj za enako stopnjo varnosti kot RSA, potrebuje krajše ključe. Primerjava dolžine ključev je prikazana v Tabeli 4.3.

Naj bo $\mathcal{O}$ dan obseg. **Eliptična krivulja** $\mathcal{E}$ je množica točk v ravnini,

$P = (x, y)$, $x, y \in \mathcal{O}$, ki zadostijo enačbi

$$y^2 = x^3 + ax + b,$$

za neki konstanti $a, b \in \mathcal{O}$. V tej kratki predstavitvi uporabe eliptičnih krivulj se omejimo na primer $\mathcal{O} = \mathbb{Z}_p$, kjer je p praštevilo, glej Jurišić in Menezes [4], in s $-P$ označimo točko $(x, -y)$. Za različni dani točki P in Q neke eliptične krivulje, $P \neq -Q$, označimo vsoto $P+Q$ s točko R. Slednja je definirana s tem, da točke P, Q in $-R$ ležijo na skupni premici. V primeru, da je $P = Q$, je vsota tista točka R za katero $-R$ leži na tangenti krivulje $\mathcal{E}$ v točki P. Za $P = (x_P, y_P)$ in $Q = (x_Q, y_Q)$ velja

$$x_R = s^2 - x_P - x_Q \quad \text{in} \quad y_R = -y_P + s(x_P - x_R),$$

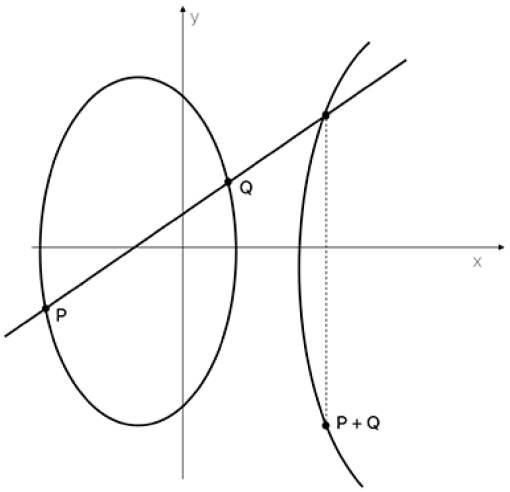
kjer je

$$s = \begin{cases} \dfrac{y_P - y_Q}{x_P - x_Q}, & \text{za } P \neq Q, \\ \dfrac{3x_P^2 + a}{2y_P}, & \text{za } P = Q. \end{cases}$$

Preostane nam še primer, ko je $P = -Q$. V tem primeru premica skozi točki P in Q ne seka $\mathcal{E}$ še v tretji točki, a se tej nevšečnosti izognemo tako, da dodamo še točko ∞ ter postavimo naslednja pravila

$$P + (-P) = \infty, \qquad P + \infty = P, \qquad \infty + P = P \quad \text{za vsak } P \in \mathcal{E}.$$

S tem postane $\mathcal{E}$ komutativna grupa, ∞ njena enota, $-P$ pa inverzen element za P. Primer seštevanja točk na eliptični krivulji je prikazan na Sliki 4.4. Izkaže se, da je v tej grupi za dani točki P in Z, kjer je $kP = Z$ za nek $k \in \mathbb{N}$, težko najti število k. Slednji problem je poznan pod imenom **diskretni logaritem** in je eksponentno težji kot npr. v grupi $\mathbb{Z}_p$. Prav to omogoča uporabo manjšega preštevila p in s tem bolj učinkovitega računanja. Zato ECC uporabljamo za kriptosisteme z javnimi ključi, konkretno za dogovor o ključu (Diffie-Hellman) ter pri digitalnih podpisih (ECDSA). Več o podrobnostih o implementaciji kriptosistemih z eliptičnimi krivuljami najdete v Menezes et. al. [6].



Slika 4.4: Prikaz seštevanja točk na krivulji.

4.5 Bluetooth

Bluetooth je brezžična tehnologija za povezovanje in izmenjavo podatkov med različnimi elektronskimi napravami. Razvilo ga je podjetje Ericsson leta 1994, sedaj pa ga upravlja SIG (Special Interest Group), ki ima več kot 30.000 članov po vsem svetu. SIG nadzoruje razvoj standardov in protokolov hkrati pa tudi skrbi za sprejemanje novih članov in ščiti blagovno znamko [15].

Osnovna ideja te tehnologije je povezovanje mobilnih naprav med seboj in z ostalimi napravami (npr. slušalke, tipkovnica, itd), na kratke razdalje, za nizko ceno in brez kablov. Bluetooth lahko uporabljajo tako bolj zmogljive naprave (npr. računalnik ali mobilni telefon), kot tudi manj zmogljive naprave (npr. slušalke). Tako so mnoge naprave tako procesorsko, spominsko, kot tudi energetsko precej omejene. Problem je tudi v tem, da mnoge naprave nimajo tipkovnice ali zaslona in premorejo komaj kakšen gumb ali led diodo.

Bluetooth naprave delujejo na frekvenčnem območju $[2.4\text{GHz}, 2.485\text{GHz}]$. Glede na moč so razdeljene v 4 razrede (Tabela 4.4). Naprave 1. razreda so

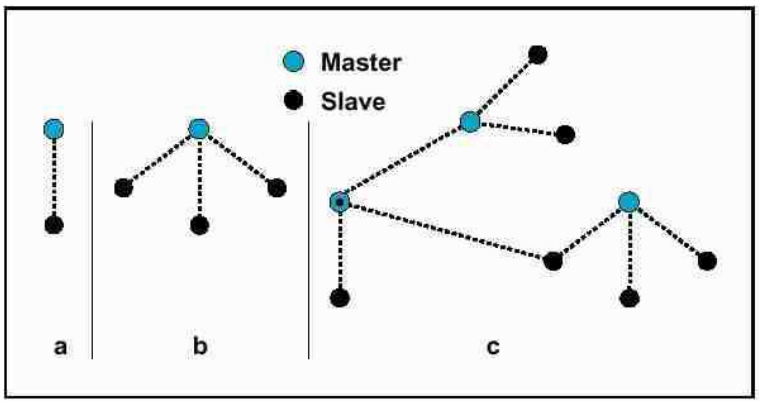
Razred	Max. dovoljena moč		Običajni doseg (m)
	(mW)	(dBm)	
1	100	20	∼100
2	2.5	4	∼10
3	1	0	∼1
4	0.5	-3	∼0.5

Tabela 4.4: Lastnosti posameznega razreda.

namenjene industrijski rabi in imajo doseg okrog 100 m, naprave 2. razreda imajo domet okrog 10 m, naprave 3. in 4. razreda pa imajo doseg okrog 1 oz. 0.5 m.

Vsaka naprava v omrežju je lahko nadrejena (ang. master) ali podrejena (ang. slave). Naprave se povežejo v celico, ki jo imenujemo *piconet*. Vsaka celica ima natančno eno nadrejeno napravo preko katere poteka vsa komunikacija med ostalimi napravami. Vsaka naprava je član vsaj ene piconet celice. Naprave, ki pripadajo več celicam hkrati, lahko med seboj virtualno povežemo. Primeri vseh načinov so prikazani na Sliki 4.5 [16]

Največja prednost Bluetooth tehnologije je njena robustnost, nizka cena, nizka poraba energije in enostavnost uporabe. Njena slabost je, da v sami zasnovi ni bilo poudarka na varnosti [2]. Raziskave so pokazale, da lahko, pri komunikaciji med dvema napravama, napadalec odkrije ključ, ki se uporablja za šifriranje podatkov. Gre za tako imenovani napad z možem v sredini (ang. man in the middle), kar pomeni, da napadalec lahko podatke bere in jih hkrati tudi aktivno spreminja [7].



Slika 4.5: Prikaz različnih načinov povezav: a) Enostavni (single slave) b) Večodjemalski (multislave) c)Kombinirano omrežje (scatternet)

Poglavje 5

Razvoj aplikacije Sef

Osrednji del našega dela je razvoj aplikacije Sef. Tako si bomo v tem poglavju podrobneje pogledali našo rešitev za varno hranjenje gesel. Začeli bomo z osnovno idejo, v kateri bomo razložili razloge za izbiro tehnologije za prenos gesel ter za izbiro mobilne platforme. Sledi opis zgradbe aplikacije. V tretjem razdelku se posvetimo delovanju aplikacije. Zaključimo s predlogi za nadaljnje izboljšave.

5.1 Ideja

Naša ideja je bila narediti prototipno/konceptno aplikacijo za varno hranjene gesel, ki za sinhronizacijo podatkov med napravami ne uporablja oblačnih storitev. Najprej smo sestavili nabor naprav, ki jih uporabljamo (npr. osebni računalnik, prenosni računalnik, tablični računalnik, pametni telefon, itd.) in premislili katero napravo imamo običajno pri sebi. Ugotovili smo, da je to običajno pametni telefon. Iz tega stališča se nam zdi to idealno mesto za hranjenje gesel. Na tem mestu je potrebno izpostaviti, da se je v zadnjem času močno povečala količina virusov in ostale zlonamerne programske kode za mobilne naprave. Poleg tega so mobilni telefoni stalno povezani z omrežjem in tako še dodatno izpostavljeni morebitnim napadom.

5.1.1 Izbira tehnologije za prenos gesel

Ker želimo do gesel dostopati na več napravah smo aplikacijo razdelili na dva dela. Prvi del se nahaja na pametnem telefonu drugi del pa na osebnem računalniku. Pri tem rabimo način za prenos podatkov iz ene naprave na drugo. Tako smo za prenos podatkov med obema napravama izbirali med dvema tehnologijama, ki sta vgrajeni v večino novejših računalnikov in pametnih telefonov:

(a) **Wi-Fi.** Ta nam omogoča hiter prenos podatkov, vendar ima tudi precej velik doseg, kar pa ni ravno najbolj primerno za našo aplikacijo.

(b) **Bluetooth.** Omogoča precej manjše hitrosti, a ima hkrati tudi manjši doseg (okrog 10 metrov), kar je po naši oceni bolj primerno.

Odločili smo se za uporabo tehnologije bluetooth v4.0 (poznane tudi pod imenom bluetooth LE). Seveda ima brezžični prenos podatkov tudi svoje slabosti. Tako je možno izvesti DOS napad na katerega bluetooth ni odporen, vendar se lahko uporabnik relativno hitro premakne iz napadalčevega dometa. Veliko večja težava je zagotoviti varnost podatkov. Ker podatki potujejo po zraku v vse smeri, je nemogoče preprečiti, da bi jih ne bi kdo prestregel. Ker bluetooth uporablja le osnovno zaščito, smo se odločili vse prenašane podatke še dodatno šifrirati.

5.1.2 Izbira mobilne platforme

Pri izbiri mobilne platforme smo se odločali med operacijskima sistemoma:

- Android in
- iOS.

Pri tem smo upoštevali več faktorjev. Predstavimo samo nekaj najpomembnejših:

(a) **Varnost.** Večino zlonamerne programske kode (ang. malware) je namenjene Androidu (Po nekaterih ocenah kar 97% [66]). Razlogov za to je več. Operacijski sistem Android je bolj odprt in tako imajo napadalci lažji dostop do podatkov. Distribucija aplikacij poteka preko različnih spletnih trgovin, ki niso nadzirane s strani Googla. Tako uporabnikom nihče ne zagotavlja, da aplikacije niso okužene s kakšnim virusom. Če za primerjavo vzamemo iOS, je tam čisto drugačen koncept. Razvijalci lahko uporabljajo samo tiste API-je, ki jih dovoli Apple. Če se programerji tega ne držijo in poskušajo zaobiti sistem, je velika verjetnost, da aplikacija ne bo odobrena in je ne bo možno objaviti na App Storu. Nadalje je tukaj še obvezen pregled aplikacije s strani Appla. Vsaka aplikacija gre pred objavo na App Storu, na obvezen pregled. Res, da se je v preteklosti že zgodilo, da so odobrili tudi kakšno aplikacijo, ki je vsebovala zlonamerno programsko kodo in tega niso odkrili, vendar je delež le-teh veliko manjši. Na koncu je tukaj še distribucija aplikacij, ki poteka samo preko App Stora in na ta način doseže širši krog uporabnikov.

(b) **Razdrobljenost.** Velika večina iOS uporabnikov ima nameščeno zadnjo verzijo operacijskega sistema in s tem tudi zadnje varnostne posodobitve. Na drugi strani Android uporabniki precej redkeje posodabljajo sistem, tako da so še dodatno izpostavljeni varnostnim tveganjem. Če primerjamo podatke iz konca novembra 2016 vidimo, da je imelo 63% uporabnikov nameščeno (zadnjo) verzijo iOS10, 29% pa iOS9 [35]. Operacijski sistem Android je le 0,4% uporabnikov uporabljalo (zadnjo) verzijo 7.x in 26,3% verzijo 6.0 [12]. Zaskrbljujoč je predvsem podatek, da skoraj 50% uporabnikov še vedno uporablja operacijski sistem, ki je starejši od dveh let.

(c) **Digitalni podpis.** Vse aplikacije, ki so objavljene na App Storu so digitalno podpisane z zasebnimi ključi razvijalcev ter zasebnim ključem Appla. Ob vsakem zagonu aplikacije na mobilnem telefonu se preveri

digitalni podpis aplikacije in če se ta ne ujema z originalnim, se aplikacija ne bo zagnala.

Po tehtnem premisleku smo se odločili, da bomo naredil aplikacijo za operacijski sistem iOS. Za drugi del aplikacije smo izbrali, operacijski sistem macOS. Pri tej odločitvi je bil zelo pomemben faktor možen način distribucije aplikacije. Tako jo lahko objavimo na App Storu in s tem zagotovimo, da bo končni uporabnik uporabljal našo aplikacijo in ne kakšen ponaredek, ki bi poskušal ukrasti podatke, saj so aplikacije ravno tako digitalno podpisane.

5.2 Zgradba

Kot je bilo že rečeno, je aplikacija sestavljena iz dveh delov. Prvi, ki ima podatkovno bazo, v kateri so varno shranjeni uporabnikovi podatki, je narejen za pametne telefone. Drugi del, namenjen dostopu do podatkov, pa je narejen za osebne računalnike.

5.2.1 Mobilna aplikacija

Mobilna aplikacija je bila razvita za operacijski sistem iOS 9.x. Razvoj je potekal v programskem okolju Xcode 7, v programskem jeziku Swift 2.3. Aplikacija je namenjena hranjenju in upravljanju podatkov ter omogoča povezavo z drugo napravo, iz katere lahko tudi upravljamo s temi podatki. Aplikacijo razdelimo na naslednje logične dele:

- gesla in zapiski,
- naprave,
- dnevnik,
- nastavitve.

Gesla in zapiski omogočajo varno hranjene gesel in zapiskov. Pri geslih je možno nastaviti ime, spletni naslov, uporabniško ime, geslo ter njegovo veljavnost. Pri kreiranju novega gesla oziroma pri spreminjanju obstoječega

je možen ročen vnos ali pa avtomatsko generiranje naključnega gesla. Uporabnik lahko nastavi veljavnost gesla, kar pomeni, da aplikacija opozarja uporabnika, da se približuje potek veljavnosti. Pri zapiskih je možen vnos imena in nekega sporočila. Tako pri geslih kot tudi zapiskih se avtomatsko beleži tudi datum, ko je bil zapis kreiran, kot tudi datum zadnje spremembe.

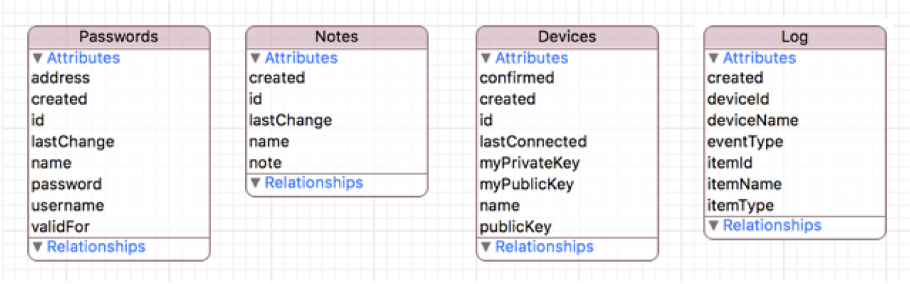
Na seznamu naprav lahko vidimo naprave s katerimi se je aplikacija povezovala v preteklosti. Pri posamezni napravi lahko vidimo njen javni ključ, ime računalnika, datum prve in zadnje povezave. Posamezno napravo je možno tudi ročno odstraniti iz seznama. Zaradi dodatne varnosti mora biti v času komunikacije z računalnikom ta stran vedno odprta na zaslonu (da ne pride do vzpostavitve povezave brez uporabnikove vednosti).

Dnevnik služi kot dodaten varnostni element, saj so v njem shranjeni vsi dogodki. Tako se beležijo dogodki kot so npr. dodajanje novega gesla, spreminjanje gesla, prikaz gesla (enako velja tudi za zapiske), vzpostavitev povezave z napravo, itd. Vsakič se zabeleži dogodek, datum dogodka, ime naprave, ki je sprožilo ta dogodek, ter element na katerega je vezan dogodek (geslo, zapisek ali naprava). Do pregleda dnevnika lahko pridemo na več načinov. Če nas zanima dnevnik za neko specifično geslo, si ga lahko ogledamo kar pri njem. Tako vidimo samo dogodke, ki so direktno povezani s tem geslom. Drug način je sledenje preko skupnega dnevnika, kjer je razviden seznam dogodkov, ne pa tudi za katero specifično geslo gre.

V nastavitvah lahko uporabnik spremeni svoje vstopno geslo (ki se uporablja za generiranje šifrirnega ključa) ter privzete nastavitve za dolžino avtomatsko generiranega gesla.

5.2.2 Aplikacija za osebne računalnike

Aplikacija je bila razvita za operacijski sistem mac osX 10.11 (El Capitan). Razvoj je potekal v programskem okolju Xcode 7, s programskim jezikom Objective-C. Aplikacija je namenjena dostopu podatkov, ki so shranjeni na pametnem telefonu. Tako lahko beremo in urejamo vsa gesla ter zapiske.



Slika 5.1: Zgradba podatkovne baze.

5.2.3 Podatkovna baza

Podatkovna baza se nahaja na mobilni aplikaciji in je relativno preprosta. Sestavljena je iz štirih tabel (prikazanih na Sliki 5.1) med katerimi pa ni nobene relacije. Tabeli za shranjevanje gesel in zapiskov sta si zelo podobni. Tako obe vsebujeta id, ime, datum kreiranja zapisa ter datum zadnje spremembe. Tabela zapiskov ima še polje za zapiske, tabela gesel pa še polja za spletni naslov, uporabniško ime ter geslo in polje za nastavljanje veljavnosti gesla.

Tabela naprav poleg id-ja in datuma kreiranja zapisa hrani še ime naprave, datum zadnje povezave, javni ključ naprave ter svoj javni in zasebni ključ. Mobilna aplikacija za vsako novo napravo generira svoj par javnega in zasebnega ključa. Tako v primeru, če pride do razkritja zasebnega ključa pri eni napravi to ne vpliva na varnost komunikacije z ostalimi napravami in ga je možno dokaj enostavno zamenjati (napravo odstranimo iz seznama in jo ponovno povežemo).

Nadalje smo vpeljali še tabelo, ki hrani zgodovino o dostopih do podatkov na mobilnem telefonu. Tako se za vsak dogodek shrani čas, id in ime naprave, ki je sprožila dogodek, id, ime in tip objekta nad katerim se je dogodek zgodil ter tip dogodka (npr. urejanje, dodajanje, itd.). Na prvi pogled bi bilo logično, da bi imela ta tabela relacijske povezave z ostalimi tabelami, vendar se pojavi vprašanje, kaj narediti, ko se nek zapis izbriše. Tako bi se lahko zgodilo, da bi z brisanjem enega gesla iz zgodovine pobrisali tudi vse

zapise o tem geslu. Druga možnost bi bila, da bi z brisanjem samo nastavili neko zastavico in potem tega zapisa ne bi več prikazovali na seznamu, na zgodovino pa to ne bi vplivalo. Težava pri tej rešitvi je, da če uporabnik želi nekaj izbrisati mu moramo to tudi dovoliti, ne pa da mu samo skrijemo.

5.3 Delovanje

Ob zagonu aplikacije se mora uporabnik najprej overiti. Postopek overjanja poteka v dveh korakih:

(a) najprej uporabnik vnese svoje geslo,

(b) nato se overi s prstnim odtisom.

Na ta način smo dosegli 2-stopenjsko overjanje uporabnika.

Ko uporabnik zaključi z vnosom svojega gesla, se v ozadju izvedeta naslednji funkciji:

(a) Iz danega gesla se izračuna njegova zgostitev, ki jo primerjamo s shranjeno zgostitvijo. Če se ujemata, je prvi del overjanja uspešen, drugače pa ne. Za izračun se uporablja zgoščevalna funkcija SHA-256, kateri dodamo tudi sol. S tem dosežemo večjo varnost, saj je napad z mavričnimi tabelami dodatno otežen (napadalec bo moral za vsakega uporabnika posebej izračunati mavrično tabelo).

(b) Druga funkcija na tem geslu izračuna varen šifrirni ključ, s katerim so šifrirani shranjeni podatki. Za izračun šifrirnega ključa se uporablja funkcija za izpeljavo ključa iz gesla (PBKDF2-SHA256) s 4096 cikli. Tak način izračuna šifrirnega ključa ščiti predvsem pred napadom z grobo silo, saj lahko napadalec preveri manj možnih ključev v enakem času, kot če ne bi uporabljali te funkcije. Šifrirni ključ se nato shrani v pomnilnik in je v njem dosegljiv dokler je aplikacija aktivna, ko gre aplikacija prvič v ozadje, se ključ izbriše in uporabnik mora ponoviti postopek.

Geslo se nastavi ob prvi uporabi aplikacije in ga je možno spremeniti v nastavitvah aplikacije.

Koda 5.1: Del programske kode, ki iz vnesenega gesla in shranjene soli generira kluč, ki se uporablja za šifriranje shranjenih podatkov

```
let password: Array<UInt8> = geslo.utf8.map {$0}
let salt: Array<UInt8> = sol.utf8.map {$0}

let key = try! PKCS5.PBKDF2(password: password, salt: salt,
  iterations: 4096, variant: .sha256).calculate()
```

Med tem, ko se v ozadju preverja geslo in izračunava šifrirni ključ, se uporabnik overi še s prstnim odtisom. Za vse v povezavi s prstnimi odtisi skrbi operacijski sistem telefona. Tako aplikacija nima neposrednega dostopa do podatkov o prstnih odtisih. Zahteva lahko le overjanje s prstnim odtisom ter na tačin dobi odgovor operacijskega sistema, ali je overjanje uspešno ali ne.

Ko se uporabnik uspešno prijavi, ima dostop do vseh podatkov v aplikaciji. Tako lahko ureja gesla, zapiske, naprave, spreminja nastavitve in pregleduje dnevnik preteklih dogodkov. Pri geslih in zapiskih so šifrirani vsi podatki razen imena in ID-ja. Za šifriranje podatkov se uporablja šifrirni algoritem AES-256 v CBC načinu.

5.3.1 Komunikacija med aplikacijama

En bolj pomembnih delov aplikacije je tudi povezava z osebnim računalnikom. Pri tem smo uporabili brezžično bluetooth komunikacijo. Ker slednja nima dovolj visoke stopnje zaščite, smo se odločili za dodatno šifriranje na podatkovnem nivoju. Tako smo za izmenjavo šifrirnega ključa med napravama uporabili kriptogsistem z javnim ključem, nato pa smo jih šifrirali s šifrirnim algoritmom AES-256. Za dogovor o ključu bi lahko uporabili digitalna potrdila, ki bi bila izdana s strani certifikatne agencije, vendar to prinese dodatne težave z upravljanjem digitalnih potrdil. Tako bi morali za preverjanje, če je neko digitalno potrdilo veljavno, imeti internetno povezavo. Poleg tega pa smo odvisni od certifikatne agencije in razpoložljivosti njihovih storitev.

Zato smo se odločili za nekoliko drugačen pristop. Aplikacija na računalniku ob prvem zagonu generira javni in zasebni ključ. Na mobilni aplikaciji se ob prvi vzpostavitvi povezave z računalnikom generira par ključev, ki se uporablja samo za ta računalnik. Tako se za vsako napravo uporablja drug par ključev. Ker želimo čim boljšo varnost, smo se odločili, da bomo uporabili kriptosistem z eliptičnimi krivuljami, ki uporablja 256-bitne ključe.

Za vzpostavitev komunikacije med napravama smo uporabili programsko ogrodje CoreBluetooth. To ogrodje nam omogoča relativno enostavno vzpostavitev komunikacije med dvema bluetooth napravama. Slabost tega ogrodja je, da smo omejeni na bluetoth LE (ang. low energy).

Preden gremo v podrobnosti komunikacije, moramo predstaviti uporabljeno terminologijo. Ko govorimo o bluetooth povezavi se namesto izrazov klient in strežnik, uporabljata izraza central in peripheral. Peripheral je naprava, ki hrani podatke (strežnik – v našem primeru je to pametni telefon). Central pa je naprava, ki dostopa do teh podatkov (klient – v našem primeru je to osebni računalnik). Poleg tega pa ima vsaka naprava seznam storitev (ang. service). Vsaka storitev ima lahko več lastnosti (ang. characteristic).

Vsaka storitev in vsaka lastnost mora imeti določen svoj unikaten id. Tako obstaja seznam standardnih id-jev [29] (npr. za storitve kot so napolnjenost baterije, ime proizvajalca, itd). V našem primeru smo naredili storitev za prenos podatkov, ki ima eno lastnost (sprejemanje in pošiljanje podatkov). Ker gre za našo, po meri narejeno storitev smo morali določiti unikaten id (tako za storitev kot tudi za lastnost). Gre za dve 128-bitni števili, ki smo ju generirali v terminalu z ukazom “uuidgen”.

Koda 5.2: Določitev unikatinh id-jev za našo storitev ter lastnost

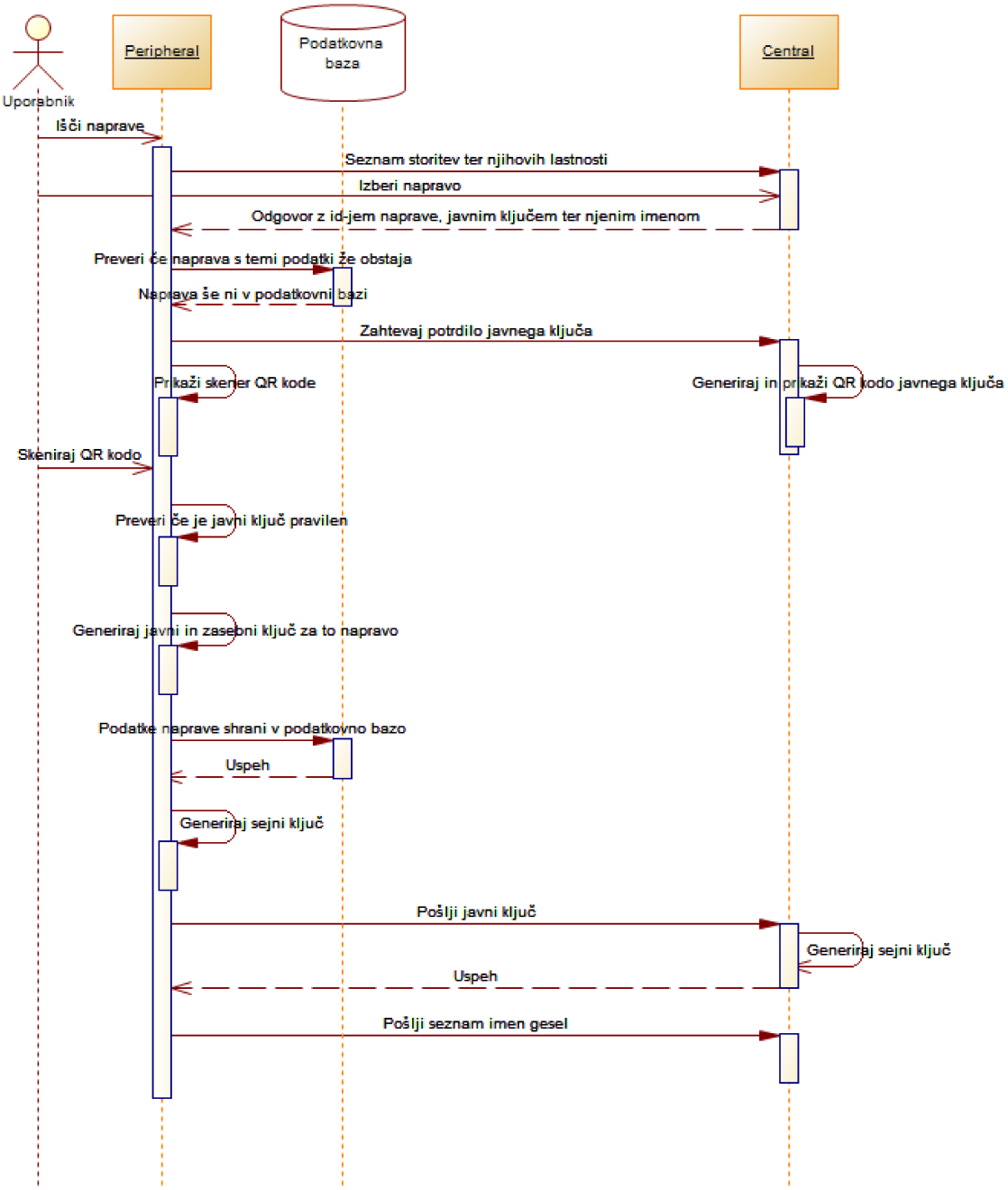
```
let transferServiceUUID = CBUUID(string: "5F833346-CF4A-415C-AA88-498A30174FD6")
let transferCharacteristicUUID =
        CBUUID(string: "7DDEE957-4861-4FB5-8F8C-5438764D2411")
```

Za vzpostavitev povezave mora najprej uporabnik ročno sprožiti storitev na mobilnem telefonu. Le-ta prične z oddajanjem seznama svojih lastnosti. Nato mora uporabnik na osebnem računalniku iz seznama naprav (tam so

samo naprave, ki podpirajo našo storitev) izbrati ustrezno napravo. Posredovanje uporabnika smo uvedli zato, da preprečimo napad, pri katerem bi nekdo dostopal do podatkov brez uporabnikove vednosti.

Ko uporabnik potrdi napravo na osebnem računalniku, central pošlje svoje osnovne podatke, kot so: id, ime ter svoj javni ključ. Ko peripheral prejme te podatke, najprej preveri, če se v bazi seznanjenih naprav že nahajajo ti podatki. Če je naprava nova, potem peripheral zahteva potrditev javnega ključa. V tem primeru central, iz svojega javnega ključa, generira in prikaže QR-kodo. Potem jo mora uporabnik prebrati z mobilnim telefonom. Na tak način se javni ključ prenese preko dveh kanalov, s čimer potrdimo, da javni ključ res prihaja iz te naprave (pri tem predpostavimo, da je aplikacija nameščena iz App Stora in ni bila spremenjena). Če se javni ključ ujema s ključem, ki je bil poslan v prvem koraku, potem se ta shrani v bazo seznanjenih naprav, nato pa peripheral pošlje svoj javni ključ. Za dogovor o sejnem ključu se uporabi protokol ECDH (Elliptic curve Diffie–Hellman) [25]. Tako se na vsaki napravi iz javnega in zasebnega ključa izračuna skupni ključ. Nato na skupnem ključu uporabimo zgoščevalno funkcijo SHA256, ki nam vrne 256 bitno zgostitev. To zgostitev pa uporabimo za sejni ključ. Na ta način je povezava dokončno vzpostavljena. Če na kateremkoli koraku pride do kakšne napake, se povezava takoj prekine in postopek je potrebno ponoviti. Celoten potek komunikacije je prikazan na Sliki 5.2

Pri pošiljanju podatkov preko povezave bluetooth naletimo na eno manjšo oviro. Protokol je bil narejen za pošiljanje manjše količine podatkov (npr. temperaturo iz senzorja ali podatek o srčnem utripu) in ne za takšne kot jih rabimo v našem primeru. Tako je priporočljivo, da je vsak poslani paket dolžine do 20 bajtov. Zaradi tega moramo podatke razrezati na bloke ustrezne dolžine in poslati vsak blok posebej. Da pa naprava na drugi strani ve, da je prejela vse bloke se vedno, kot zadnji blok, pošlje oznaka “EOM”.



Slika 5.2: Komunikacijski diagram, ki prikazuje vzpostavitev prve povezave med napravama.

Koda 5.3: Funkcija, ki podatke razdeli na posamezne bloke in jih pošlje napravi na drugi strani

```
let NOTIFY_MTU = 20
private var dataToSend: NSData?
private var sendDataIndex: Int?
private var sendingEOM = false

private func sendData() {

  //if end of message
  if(sendingEOM){
    let didSend = peripheralManager?.updateValue(
      "EOM".dataUsingEncoding(NSUTF8StringEncoding)!,
      forCharacteristic: responseCharacteristic!, onSubscribedCentrals: nil)

        if(didSend == true){
          //data sent
          sendingEOM = false
        }
        return
      }

  // We're not sending an EOM, so we're sending data
  // Is there any left to send?
  guard sendDataIndex < dataToSend?.length else{
    // No data left.  Do nothing
    return
  }

  // There's data left, so send until the callback fails, or we're done.
  var didSend = true

  while(didSend){
    // Work out how big it should be
    var amountToSend = dataToSend!.length - sendDataIndex!;

    // Can't be longer than 20 bytes
    if(amountToSend > NOTIFY_MTU){
      amountToSend = NOTIFY_MTU
    }

    // Copy out the data we want
    let chunk = NSData(bytes: dataToSend!.bytes + sendDataIndex!,
      length: amountToSend)

    // Send it
    didSend = peripheralManager!.updateValue(chunk, forCharacteristic:
      responseCharacteristic!, onSubscribedCentrals: nil)

    // If it didn't work, drop out and wait for the callback
    if(!didSend){
      return
    }

    let stringFromData = NSString(data: chunk, encoding: NSUTF8StringEncoding)

    // It did send, so update our index
    sendDataIndex! += amountToSend;

    // Was it the last one?
    if (sendDataIndex! >= dataToSend!.length){

      // It was - send an EOM
      // Set this so if the send fails, we'll send it next time
      sendingEOM = true

      // Send it
      let eomSent = peripheralManager!.updateValue(
        "EOM".dataUsingEncoding(NSUTF8StringEncoding)!,
        forCharacteristic: responseCharacteristic!, onSubscribedCentrals: nil)

      if (eomSent) {
        // It sent, we're all done
        sendingEOM = false
      }
      return
    }
  }
}
```

5.4 Uporabljene knjižnice

Pri razvoju aplikacije smo uporabili tudi tri odprtokodne knjižnice. Dve knjižnici vsebujeta implementacije različnih kriptografskih funkcij, tretja pa se uporablja za generiranje QR kod.

CryptoSwift

Knjižnica je napisana v programskem jeziku Swift in vsebuje implementacije različnih kriptografskih algoritmov. Tako vsebuje implementacije najbolj pogostih zgoščevalnih funkcij (MD5, SHA-1, SHA-2 in SHA-3), simetrične šifre (AES, Blowfish, ...), funkcije za izpeljavo ključa iz gesla (PBKDF1 in PBKDF2), itd. Knjižnica je dosegljiva na [21]

Elliptic Curve Crypto

Knjižnica je napisana v programskem jeziku Objective-C in vsebuje implementacije algoritmov ECDSA in ECDH. Knjižnica omogoča generiranje parov ključev, digitalno podpisovanje sporočil ter preverjanje pravilnosti podpisa in generiranje sejnega ključa z algoritmom ECDH. Knjižnica je dosegljiva na [24]

ZXingObjC

ZXingObjC knjižnica je napisana v programskem jeziku Objective-C in je namenjena generiranju različnih črtnih in QR kod. Omogoča tudi dekodiranje podatkov zapisanih v taki kodi. Mi smo jo uporabili za generiranje QR kode javnega ključa klienta. Knjižnica je dosegljiva na [64]

5.5 Možne nadgradnje

Tako na varnostnem področju kot tudi na izboljšanju uporabniške izkušnje so možne nadgradnje aplikacije. Na področju varnosti bi lahko uporabili še tretjo napravo za identifikacijo. Tako bi lahko uporabili pametno uro, na

kateri bi imeli shranjen žeton. Tako bi v fazi overjanja uporabnika na mobilni aplikaciji, pridobili žeton iz pametne ure (vzpostavili bi varno povezavo tako kot med pametnim telefonom in osebnim računalnikom), ter tako zagotovili 3-stopenjsko overjanje (nekaj kar uporabnik ve - geslo, nekaj kar uporabnik je - prstni odtis in nekaj kar uporabnik ima - žeton). Ta žeton bi lahko potem združili z uporabnikovim geslom in rezultat uporabili za šifriranje shranjenih podatkov. Na tak način bi napadalec moral pridobiti oba podatka, npr. samo geslo mu ne bi nič pomagalo. Dodatno bi lahko na pametni uri hranili še druge žetone, katere bi uporabili za generiranje gesel. Tako bi imeli na pametnem telefonu en žeton, na pametni uri drug žeton in ko bi ju združili, bi dobili geslo. Na tak način, dejansko geslo ne bi bilo shranjeno na nobeni napravi.

Za izboljšanje uporabniške izkušnje bi bilo dobro naredi razširitve za najbolj pogoste spletne brskalnike. Tako bi uporabnik moral samo vzpostaviti povezavo med vsemi napravami in potem bi se mu gesla, v prijavnih obrazcih avtomatsko izpolnjevala. Poleg tega bi bilo dobrodošlo narediti tudi razvoj aplikacije za ostale platforme (Android, Windows Phone, Windows, Linux, itd.).

Trenutno se vsi podatki avtomatsko shranijo v varnostno kopijo operacijskega sistema (če ima uporabnik vključeno avtomatsko izdelavo varnostne kopije). Kljub temu, bi bilo dobro dodati možnost izdelave varnostne kopije podatkovne baze. Na tak način bi lahko podatke prenesli tudi na aplikacijo na drugi platformi.

Poglavje 6

Sklepne ugotovitve

V magistrskem delu so predstavili osnove overjanja. Pogledali smo si različne načine ter pri tem omenili njihove prednosti in slabosti. Videli smo, da uporabniki še vedno uporabljajo slaba gesla in da za več sistemov uporabljajo ista gesla. Vendar težava niso vedno samo uporabniki, temveč tudi razvijalci in skrbniki sistemov (npr. spletnih strani), ki slabo skrbijo za varnost uporabnikovih podatkov oz. ne obveščajo svojih uporabnikov, kadar pride do kakšnih varnostnih incidentov oz. vdorov v njihov sistem.

Predstavili smo nekaj rešitev za upravljanje z gesli ter njihove pristope. Pri tem smo videli nekaj dobrih praks, kjer se trudijo, da so njihovi sistemi čim bolj varni in da je matematično dokazano, da njihovi zaposleni ne morejo dostopati do uporabnikovih podatkov.

Cilj magistrske naloge je bil izdelati aplikacijo, ki bi omogočala varno hranjenje gesel. Pri tem smo želeli, da so gesla vedno pod uporabnikovim nadzorom (vedno so shranjena na napravi, ki jo ima pri sebi) in vedno dosegljiva. Za razliko od običajnih aplikacij za hranjenje gesel, kjer za sinhronizacijo podatkov uporabljajo storitve v oblaku, smo mi izbrali drug pristop. Tako so gesla shranjena na pametnem mobilnem telefonu in do njih dostopamo preko bluetooth povezave.

Literatura

[1] N. Courtois in J. Pieprzyk, Cryptanalysis of Block Ciphers with Overdefined Systems of Equations, v zborniku: ASIACRYPT 2002, LNCS 2501, str. 267–287.

[2] B. K. Mandal, D. Bhattacharyya in T. Kim, A Design Approach for Wireless Communication Security in Bluetooth, *International J. Security and its Appl.* **8**/2 (2014), str. 341.

[3] A. Menezes, P. van Oorschot in S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[4] A. Jurišić in A. Menezes, Elliptic Curves and Cryptography, *Dr. Dobb's Journal* (1997/April), 26–37.

[5] S. Landau, Communication Security for the Twenty-first Century, *Notices of the AMS* **47**/4 (2000), 450–459.

[6] A. Menezes, D. Hankerson in S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer, 2004.

[7] N. Minar in M. Tarique, Bluetooth Security Threats and Solutions: A survey, *Internat. J. Distributed and Parallel Systems* (IJDPS) **3**/1 (2012), 127–148.

[8] D. R. Stinson, *Cryptography – Theory and Practice*, CRC Press, 3. izdaja, 2006.

[9] C. P. Pfleeger in S. L. Pfleeger, Security in Computing, 5. izdaja, Prentice Hall, 2015.

[10] J. Walker, M. E. Kounavis, S. Gueron in G. Graunke, Recent Contributions to Cryptographic Hash Functions, *Intel Tech. J.* **13** (2009), str. 80–95. (tudi *Dr. Dobb's Journal*, 2009/Aug.)

Internetni viri

[11] Announcing the ADVANCED ENCRYPTION STANDARD (AES) [Online, 16.12.2016]. Dostopno na: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[12] Android (operating system) [Online, 16.12.2016]. Dostopno na: https://en.wikipedia.org/wiki/Android_(operating_system)

[13] Android's Trust API: a short history, and why it's a game changer [Online, 1.12.2016]. Dostopno na: https://thisdata.com/blog/androids-trust-api-a-short-history-and-why-its-a-game-changer/

[14] Apple, iOS Security, 2015 [Online, 1.09.2016]. Dostopno na: https://www.apple.com/business/docs/iOS_Security_Guide.pdf

[15] Bluetooth [Online, 16.12.2016]. Dostopno na: https://en.wikipedia.org/wiki/Bluetooth

[16] Bluetooth Baseband [Online, 16.12.2016]. Dostopno na: http://ecee.colorado.edu/~ecen4242/marko/Bluetooth/Bluetooth/SPECIFICATION/Baseband.htm

[17] Court rules: Touch ID is not protected by the Fifth Amendment but Passcodes are [Online, 1. 12. 2016]. Dostopno na: https://www.engadget.com/2014/10/31/court-rules-touch-id-is-not-protected-by-the-fifth-amendment-bu/

[18] Cryptanalysis of MD5 and SHA: Time for a New Standard [Online, 16.12.2016]. Dostopno na: https://www.schneier.com/essays/archives/2004/08/cryptanalysis_of_md5.html

[19] Cryptanalysis of SHA-1 [Online, 16.12.2016]. Dostopno na: https://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html

[20] Cryptographic hash function [Online, 16.12.2016]. Dostopno na: https://en.wikipedia.org/wiki/Cryptographic_hash_function

[21] CryptoSwift [Online, 16.12.2016]. Dostopno na: https://github.com/krzyzanowskim/CryptoSwift

[22] D. Danchev, Survey: 60 percent of users use the same password across more than one of their online accounts [Online, 1. 12. 2015]. Dostopno na: http://www.zdnet.com/article/survey-60-percent-of-users-use-the-same-password-across-more-than-one-of-their-online-accounts/

[23] DES [Online, 16.12.2016]. Dostopno na: https://sl.wikipedia.org/wiki/DES

[24] Elliptic Curve Crypto [Online, 16.12.2016]. Dostopno na: https://github.com/ricmoo/GMEllipticCurveCrypto

[25] Elliptic curve Diffie–Hellman [Online, 16.12.2016]. Dostopno na: https://en.wikipedia.org/wiki/Elliptic_curve_Diffie\OT1\textendashHellman

[26] FIPS 202 [Online, 16.12.2016]. Dostopno na: http://csrc.nist.gov/publications/drafts/fips-202/fips_202_draft.pdf

[27] FIPS 180-3 [Online, 16.12.2016]. Dostopno na: http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf

[28] Florida court rules police can demand your phone's passcode [Online, 16. 12. 2016]. Dostopno na: https://www.engadget.com/2016/12/14/florida-court-rules-police-can-demand-your-phones-passcode/

[29] GATT Services [Online, 16.12.2016]. Dostopno na: https://www.bluetooth.com/specifications/gatt/services

[30] Google plans to bring password-free logins to Android apps by year-end [Online, 1.12.2016]. Dostopno na: https://techcrunch.com/2016/05/23/google-plans-to-bring-password-free-logins-to-android-apps-by-year-end/

[31] Google aims to kill passwords with project Abacus [Online, 1.12.2016]. Dostopno na: https://threatpost.com/google-aims-to-kill-passwords-with-project-abacus/118288/

[32] W. Gordon, How Your Passwords Are Stored on the Internet, 2012 [Online, 1. 12. 2015]. Dostopno na: http://lifehacker.com/5919918/how-your-passwords-are-stored-on-the-internet-and-when-your-password-strength-doesnt-matter

[33] Hash function [Online, 16.12.2016]. Dostopno na: https://en.wikipedia.org/wiki/Hash_function

[34] Imperva, Consumer Password Worst Practices, White paper, 2014 [Online, 1.12.2015]. Dostopno na: http://www.imperva.com/docs/wp_consumer_password_worst_practices.pdf

[35] iOS [Online, 16.12.2016]. Dostopno na: https://en.wikipedia.org/wiki/IOS

[36] KeePass lastnosti [Online, 1.11.2016]. Dostopno na: http://keepass.info/features.html

[37] KeePass varnost podatkov [Online, 1.11.2016]. Dostopno na: http://keepass.info/help/base/security.html

[38] KeePass varnostni incidenti [Online, 1.11.2016]. Dostopno na: http://keepass.info/help/kb/sec_issues.html

[39] Key derivation function [Online, 16.12.2016]. Dostopno na: https://en.wikipedia.org/wiki/Key_derivation_function

[40] LastPass lastnosti [Online, 1.11.2016]. Dostopno na: https://lastpass.com/features/

[41] LastPass, kako deluje [Online, 1.11.2016]. Dostopno na: https://lastpass.com/how-it-works/

[42] LastPass varnost podatkov [Online, 1.11.2016]. Dostopno na: https://lastpass.com/support.php?cmd=showfaq&id=6926

[43] LastPass napad z ribarjenjem [Online, 1.11.2016]. Dostopno na: https://lastpass.com/support.php?cmd=showfaq&id=10072

[44] LastPass varnostno obvestilo [Online, 1.11.2016]. Dostopno na: https://blog.lastpass.com/2011/05/lastpass-security-notification.html/

[45] LastPass varnostno obvestilo [Online, 1.11.2016]. Dostopno na: https://blog.lastpass.com/2015/06/lastpass-security-notice.html/

[46] MD5 [Online, 1. 12. 2016]. Dostopno na: https://en.wikipedia.org/wiki/MD5

[47] NIST declares the age of SMS-based 2-factor authentication over [Online, 16.12.2016]. Dostopno na: https://techcrunch.com/2016/07/25/nist-declares-the-age-of-sms-based-2-factor-authentication-over/

[48] NIST Releases SHA-3 Cryptographic Hash Standard [Online, 16.12.2016]. Dostopno na: https://www.nist.gov/news-events/news/2015/08/nist-releases-sha-3-cryptographic-hash-standard

[49] Password app developer overlooks security hole to preserve ads [Online, 1.11.2016]. Dostopno na: https://www.engadget.com/2016/06/04/keepass-wont-fix-security-hole-due-to-ads/

[50] Passware Kit Forensic [Online, 1. 12. 2016]. Dostopno na: https://www.passware.com/kit-forensic/

[51] A. Peslyak, John the Ripper [Online, 1. 12. 2016]. Dostopno na: http://www.openwall.com/john/

[52] Report on the Development of the Advanced Encryption Standard (AES) [Online, 16.12.2016]. Dostopno na: http://csrc.nist.gov/archive/aes/round2/r2report.pdf

[53] RFC3174 [Online, 16.12.2016]. Dostopno na: https://tools.ietf.org/html/rfc3174

[54] RFC2898 [Online, 16.12.2016]. Dostopno na: https://tools.ietf.org/html/rfc2898

[55] RSA SecurID [Online, 16.12.2016]. Dostopno na: https://en.wikipedia.org/wiki/RSA_SecurID

[56] SHA-1 [Online, 16.12.2016]. Dostopno na: https://en.wikipedia.org/wiki/SHA-1

[57] SHA-2 [Online, 16.12.2016]. Dostopno na: https://en.wikipedia.org/wiki/SHA-2

[58] The Keccak sponge function family [Online, 16.12.2016]. Dostopno na: http://keccak.noekeon.org

[59] Worst, most common passwords for the last 5 years [Online, 1. 12. 2016]. Dostopno na: http://www.computerworld.com/article/3024404/security/worst-most-common-passwords-for-the-last-5-years.html

[60] L. Xing, X. Bai, T. Li,X. Wang, K. Chen, X. Liao, Unauthorized Cross-App Resource Access on MAC OS X and iOS, https://arxiv.org/abs/1505.06836

[61] Yahoo Announces It Was Hacked Again, Over 1 Billion Accounts Compromised [Online, 16. 12. 2016]. Dostopno na: http://www.iclarified.com/58323/yahoo-announces-it-was-hacked-again-over-1-billion-accounts-compromised

[62] Yahoo confirms new security breach affecting over one billion accounts [Online, 16. 12. 2016]. Dostopno na: https://www.engadget.com/2016/12/14/yahoo-confirms-a-new-security-breach-affecting-1-billion-account/

[63] Yahoo confirms over 500 million users affected in 2014 breach [Online, 1. 12. 2016]. Dostopno na: https://www.engadget.com/2016/09/22/yahoo-confirms-over-500-million-users-affected-in-2014-breach/

[64] ZXingObjC [Online, 16.12.2016]. Dostopno na: https://github.com/TheLevelUp/ZXingObjC

[65] G. Žagar, Analiza Gesel, 2010 [Online, 1. 12. 2015]. Dostopno na: http://infosec.si/?p=169

[66] 97% of malicious mobile malware targets Android [Online, 16.12.2016]. Dostopno na: https://www.scmagazineuk.com/updated-97-of-malicious-mobile-malware-targets-android/article/535410/